中国电子教育学会高教分会推荐
高等学校电子信息类“十三五”规划教材



- 随书提供PPT课件、设计案例、汇编程序源代码等资料。
- 内容精练、重点突出，强调编程思想和逻辑思维的训练。

单片机原理及应用

李桂林　王新屏　马　驰　张春光　编著

西安电子科技大学出版社
http://www.xduph.com

中国电子教育学会高教分
高等学校电子信息类“十

单片机原理及应用

李桂林　王新屏　马驰　张春光　编著

西安电子科技大学出版社

内 容 简 介

本书以MCS-51系列单片机原理和应用为主线，重点介绍单片机的结构、指令系统、汇编程序设计、内部标准功能单元、硬件系统扩展等内容，并精心设计了大量例题和多种解题思路。本书教学重点突出，叙述准确精练，完全可以满足教师课堂教学和学生课程学习之需要。

全书结构规范、系统性强、实例丰富，既注重基础知识的讲解和逻辑思维的训练，又突出工程实践和实际应用。为了方便教师教学和学生自学，随书提供PPT课件、汇编程序源代码等辅助学习资料(扫描二维码即可)。

本书既可作为普通高等院校通信工程、电子信息、自动化、电气工程、机电一体化、测控技术和仪器仪表等专业的教材，也可作为电子设计、开发爱好者的参考书。

图书在版编目(CIP)数据

单片机原理及应用/李桂林等编著.
—西安：西安电子科技大学出版社，2017.8(2017.9重印)
高等学校电子信息类"十三五"规划教材
ISBN 978-7-5606-4592-6

Ⅰ.①单… Ⅱ.①李… Ⅲ.①单片微型计算机—教材 Ⅳ.①TP368.1

中国版本图书馆CIP数据核字(2017)第174376号

策　　划　李惠萍
责任编辑　王　瑛
出版发行　西安电子科技大学出版社(西安市太白南路2号)
电　　话　(029)88242885　88201467　　邮　　编　710071
网　　址　www.xduph.com　　电子邮箱　xdupfxb001@163.com
经　　销　新华书店
印刷单位　陕西利达印务有限责任公司
版　　次　2017年8月第1版　2017年9月第2次印刷
开　　本　787毫米×1092毫米　1/16　印张 17
字　　数　402千字
印　　数　501～3500册
定　　价　34.00元
ISBN 978-7-5606-4592-6/TP

XDUP 4884001-2

前　言

一、学习单片机的意义

传统的电子系统所完成的一切功能都是通过布线逻辑控制(Wired Logic Control)实现的，若要增加功能或者改进性能，必须修改或者重新设计硬件电路。而现代电子系统完成的许多功能是利用单片机通过存储程序控制(Stored Program Control)实现的，也就是控制功能是通过计算机执行预先存储在存储器中的程序来实现的。如果想要给系统增加功能或者改进性能，只需要改写程序(即软件)即可轻而易举地达到目的，操作非常灵活。若将传统的电子系统当作一个僵死的电子系统，那么智能化的现代电子系统则是一个具有生命的电子系统。单片机应用系统的硬件结构给予电子系统以“身躯”，单片机应用系统的应用程序则赋予其“灵魂”。电子系统的智能化程度是无止境的，常常不需增添硬件资源就能实现各种功能的更新和增添，这也是当前许多电子设备的功能大量增强和不断扩展的重要因素。

单片机是一种面向控制的大规模集成电路芯片，具有体积小、性价比高、功能强、性能稳定、控制灵活等优点，已成为电子系统中最重要的智能化核心部件，是微型计算机的一个重要分支。目前，单片机技术在通信、电子信息、工业检测控制、机电一体化、电力电子、智能仪器仪表、汽车电子等领域得到了广泛的应用。其中 MCS－51 系列单片机以其特有的简单、易学、易用和高性价比的优势，占有单片机市场的大部分份额，是初学者学习单片机的首选机型。为了帮助本科生和科技人员尽快掌握单片机的基本知识和应用开发方法，在理论方面打好基础，在应用方面快速上手，编著者特精心编写了本书。

二、本书的结构和特点

参加本书编写的人员均为长期从事单片机技术教学的一线教师，具有丰富的教学经验，同时这些教师均参加过智能化电子产品的开发和研制。本书是编者依据多年来单片机课程教学和应用系统开发的经验，并参考了大量的同类书籍和单片机发展的最新技术资料编写而成的。

本书以课堂教学和课堂学习为主线，力图解决困扰大多数学生的单片机学习问题(如：基本概念理解困难，没有编程思路，很难建立中断的概念，不了解单片机软硬件之间的关系等)，从精简内容、突出重点、加强逻辑思维能力训练等方面入手编写，具有如下特点：

(1) **内容精练、重点突出**。本书缩减了一些次要内容，突出学习重点，关键内容和知识点在字体上都做了加黑处理，使得学生在学习过程中可更好地抓住重点。在指令的运用上，突出常用指令(在附录中加了“＊”)，而且所有的例题均限定在这些指令之内。

(2) **强调编程思想和逻辑思维的训练**。通过多年教学经验总结，笔者认为很大一部分学生缺乏分析问题的能力和逻辑思维能力，需要加强这方面的训练。本书以此作为切入

点，重点展开论述。针对本课程实践性强的特点，笔者对书中例题进行了精心设计，并突出了例题的多角度讲解，给出不同思路下的解题(编程)方法，以激发学生思维能力。

(3) **注重实用、与时俱进**。根据单片机外围芯片发展的实际情况，增加了串行总线技术及常用芯片的讲解，如 I^2C 总线、SPI 总线，将 AT89 系列单片机作为实例，以适应当代主流单片机和外围接口器件的发展趋势。

本书由李桂林、王新屏、马驰和张春光编写，李桂林编写第 1 章、第 4 章、第 8 章、第 9 章，并负责全书的组织和统稿工作；王新屏编写第 3 章、第 5 章及第 6 章的部分内容；马驰编写第 2 章、第 7 章的部分内容；张春光编写第 6 章的部分内容、第 7 章的部分内容。特别感谢西安电子科技大学出版社李惠萍编辑对本书编写所提出的宝贵意见，同时对本书所列参考文献的作者也在此表示诚挚的谢意。

按照编写目标，编著者进行了许多思考和努力。由于编著者水平有限，书中难免有不妥之处，恳请读者批评指正，以便不断改进(联系邮箱：modulation@sina.com)。

针对本书内容和其他扩展内容，我们开发了一个极具特色的网站：www.mcs-51.com(支持手机版)，里面有知识点的形象讲解、扩展内容和芯片的学习、C51 开发语言的学习，还有一些电子制作的实际案例讲解、电路图和全部代码，欢迎读者登录网站查阅和学习。读者也可通过扫描本页或封底的二维码下载并安装手机 App 应用。



编著者

2017 年 4 月

目 录

第 1 章　单片机基础知识

本章主要介绍单片机的基本概念、特点及应用领域，重点讲述单片机应用系统和开发语言的基本知识、常用单片机系列和分类。

1.1　单 片 机 概 述

单片机是一种集成电路芯片，伴随着微电子技术的发展而产生，是微型计算机的一个重要分支。

现代电子系统的基本核心是嵌入式计算机应用系统(简称嵌入式系统，Embedded System)，而单片机就是最典型、最广泛、最普及的嵌入式计算机应用系统，也可以称其为基本嵌入式系统。

1.1.1　单片机的组成

单片机是把中央处理器(CPU)、随机存储器(RAM，一般用于存储数据)、只读存储器(ROM，一般用于存储程序)、中断系统、定时器/计数器以及 I/O 接口电路(可能还包括显示驱动电路、脉宽调制电路、模拟多路转换器、A/D 转换器等电路)等集成在一块芯片上的微型计算机。换一种说法，单片机就是不包括输入/输出设备、不带外部设备的微型计算机。虽然单片机只是一个芯片，但从组成和功能上看，它已具有了计算机系统的属性，因此称它为单片微型计算机(Single Chip Micro - Computer，SCMC)，简称单片机。

目前，单片机已有几十个系列，上千个品种。图 1 - 1 为某些型号的单片机。在众多产品中，20 世纪 80 年代 Intel 公司推出的 MCS - 51 系列单片机应用最为广泛。

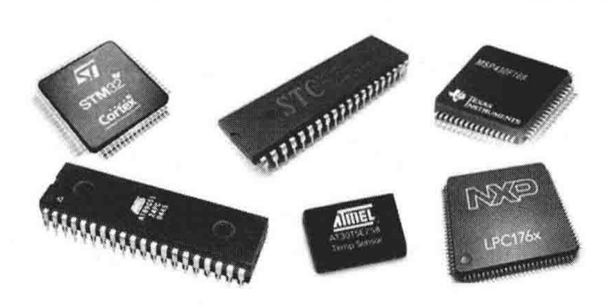


图 1 - 1　各种型号的单片机

虽然单片机型号各异，但其基本组成部分相似。图 1 - 2 为单片机的典型结构框图。

单片机在应用时通常处于被控系统的核心地位并融入其中，即以嵌入的方式使用。为了强调其"嵌入"的特点，也常常将单片机称为嵌入式微控制器(Embedded Micro-Controller Unit，EMCU)，在单片机的电路和结构中有许多嵌入式应用的特点。

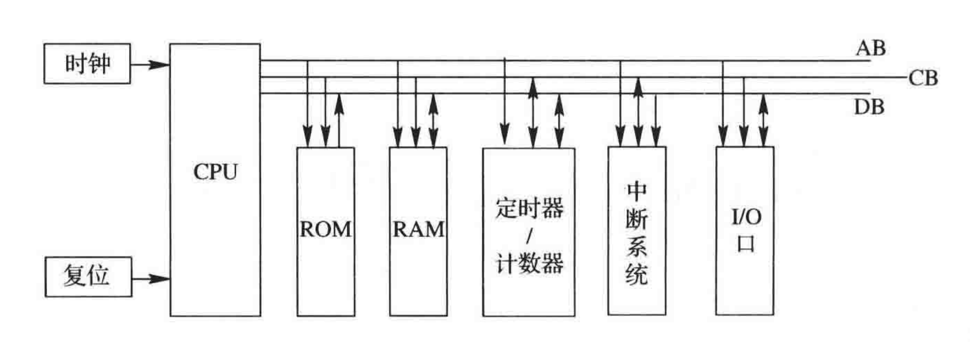


图 1－2　单片机的典型结构框图

1.1.2　单片机的特点

单片机是一种集成电路芯片，在工业控制领域得到了广泛应用。单片机的主要特点如下所述。

1. 集成度高、体积小、可靠性高

单片机将各功能部件集成在一块集成电路芯片上，所以集成度很高，体积自然也是最小的。芯片本身是按工业测控环境要求设计的，内部布线很短，数据在传送时受干扰的影响较小，其抗工业噪声性能优于一般通用的 CPU。单片机程序指令、常数及表格等固化在 ROM 中不易破坏，许多信号通道均在一个芯片内，故可靠性较高。

2. 控制功能强

为了满足实际控制要求，各类单片机的指令系统均有极丰富的条件分支转移能力、I/O口的逻辑操作及位处理能力，单片机的位操作能力更是其他计算机无法比拟的。单片机的实时控制功能特别强，非常适用于专门的控制系统。

3. 低电压、低功耗，便于生产便携式产品

为了满足广泛使用的便携式产品的开发，许多低功耗单片机的工作电压仅为 1.8～3.6V，而工作电流仅为数百微安，能够使系统在低功耗状态下运行。

4. 易扩展

单片机芯片内具有计算机正常运行所必需的部件，芯片外部有供扩展用的三总线及并行、串行输入/输出引脚，很容易构成各种规模的单片机应用系统。

5. 性能价格比优异

为了提高运行速度和工作效率，高端单片机已开始使用 RISC 流水线和 DSP 等技术。寻址能力也已突破 64 KB(B 为 Byte 的简写，即字节，为 8 位二进制码)的限制，有的已达到 16 MB，片内 RAM 容量则可达 2 MB。由于单片机的广泛使用，因而其销量极大。各大公司的商业竞争更使其价格十分低廉，所以其性能价格比极高。

1.1.3　单片机系统

初学者在学习单片机时，应注意区分单片机和单片机系统、单片机应用系统和单片机开发系统。

1. 单片机和单片机系统

单片机只是一个芯片，而单片机系统则是在单片机芯片的基础上扩展其他电路或芯片

构成的具有一定应用功能的计算机系统。

通常所说的单片机系统都是为实现某一应用需要而由开发人员设计的，是一个围绕单片机芯片而组建的计算机应用系统。在单片机系统中，单片机处于核心地位，是构成单片机系统的硬件和软件基础。

2. 单片机应用系统和单片机开发系统

单片机应用系统(简称单片机系统)主要是为应用而设计开发的，该系统与控制对象结合在一起工作，是单片机开发应用的成果。单片机系统的设计开发包括硬件设计和软件编程两部分。由于软/硬件资源所限，单片机与微型计算机不同，单片机系统本身不能实现自我开发，要进行系统设计开发，必须使用专门的单片机开发系统。

单片机开发系统是单片机应用系统开发调试工具的总称。在线仿真器(In-Circuit Emulator，ICE)是单片机开发系统的核心部分(参见9.1.2节)。在单片机系统的设计中，仿真器应用的范围主要集中在对程序的仿真上。因为，在单片机的开发过程中，程序设计是最重要的，但也是难度最大的。一种最简单和原始的开发流程是：编写程序→烧写芯片→验证功能，这种方法对于简单系统是可以应付的，但在复杂系统中使用这种方法则是完全不可能的，所以需要使用单片机开发系统来支持开发工作。

1.1.4 单片机程序设计语言

程序实际上是一系列计算机指令的有序集合。我们把利用计算机指令系统来合理地编写出解决某个问题的程序的过程，称为程序设计。这也是我们学习这门课程的主要目的之一。

单片机程序设计语言，主要是指在开发系统中使用的语言。在单片机开发系统中主要使用汇编语言和高级语言，而单片机应用系统运行时使用机器语言。

1. 汇编语言

汇编语言是用助记符表示的机器指令。汇编语言是对机器语言的改进，是单片机最常用的程序设计语言之一。汇编指令和机器指令一一对应，所以用汇编语言编写的程序效率高，占用存储空间小，运行速度快，因此汇编语言能编写出最优化的程序。虽然汇编语言是高效的计算机语言，但它是面向机器的低级语言，不便于记忆和使用，且与单片机硬件关系密切，这就要求程序设计人员必须精通单片机的硬件系统和指令系统。每一类单片机都有它自己的汇编语言，它们的指令系统是各不相同的，也就是说，不同的单片机有不同的指令系统。尽管目前已有不少程序设计人员使用C语言来进行单片机的应用程序开发，但是在对程序运行空间和时间要求很高的场合，汇编语言仍是必不可少的。

2. C语言

也可以使用高级语言进行单片机应用系统开发，最常用的是C语言。**单片机开发用的C语言是在标准C基础上经过扩充的C语言，也称为C51语言。**与汇编语言相比，C语言不受具体“硬件”的限制，具有通用性强，直观、易懂、易学，可读性好等优点。目前多数的单片机开发者使用C语言来进行程序设计。C语言已经成为人们公认的高级语言中高效、简洁而又贴近单片机硬件的编程语言。用C语言进行单片机的软件开发，可大大缩短开发周期，且可明显地增加软件的可读性，便于改进和补充。

1.2 单片机的历史和发展

单片机作为一种面向测控的微控制器，应用极为广泛。自 20 世纪 70 年代以来历经 4 位机、8 位机、16 位机、32 位机等发展过程，现已有 50 多个系列，上千个品种，新的系列和型号还不断出现，但 8 位通用单片机一直是市场上的主流。

1.2.1 单片机的发展历史

1. 单片机形成阶段

1976 年，Intel 公司推出了 MCS－48 系列单片机，这是第一个 8 位单片机。它是 8 位 CPU、1KB ROM、64B RAM、27 根 I/O 线和 1 个 8 位定时器/计数器等集成于一块半导体芯片上的单片结构。

其特点是：存储器容量较小，寻址范围小(不大于 4 KB)，无串行接口，指令系统功能不强。

这一阶段的单片机产品还有 Motorola 公司的 6801 系列和 Zilog 公司的 Z8 系列。

2. 性能完善提高阶段

1980 年，Intel 公司又推出了内部功能单元集成度更强的 8 位机——MCS－51 系列产品。其性能大大超过了 MCS－48 系列产品，一经问世便显示出其强大的生命力，广泛应用于电子信息、工业控制、仪器仪表等领域。

其特点是：结构体系完善，性能卓越，面向控制的特点进一步突出。

现在，MCS－51 已成为公认的单片机经典机种。

3. 微控制器化形成阶段

1982 年，Intel 推出 MCS－96 系列单片机。芯片内集成有 16 位 CPU、8 KB ROM、232B RAM、5 个 8 位并口、1 个全双工串行口、2 个 16 位定时器/计数器，寻址范围为 64 KB，片上还有 8 路 10 位 ADC、1 路 PWM 输出及高速 I/O 部件等。

其特点是：片内增强了面向测控系统的外围电路，使单片机可以方便灵活地用于复杂的自动测控系统及设备。

这一阶段，“微控制器(MCU)”的称谓更能反映单片机的本质。

4. 微控制器化完善阶段

近期推出的单片机产品，内部集成有高速 I/O 口、ADC、PWM、WDT 等部件，并在低电压、低功耗、串行扩展总线、控制网络总线和开发方式(在系统可编程，In System Programmable，ISP)等方面都有了进一步的增强。

其特点是：适合不同领域要求的各种通用单片机系列和专用型单片机得到了大力发展，单片机的综合品质(如成本、性能、体系结构、开发环境、供应状态)有了长足的进步。

8 位单片机从 1976 年公布至今，其技术已有了很大的发展，目前乃至将来仍是单片机的主流机型之一。

1.2.2 单片机的发展趋势

1. 低功耗

HCMOS 工艺出现后，HCMOS 器件得到了飞速的发展。如今，数字逻辑电路、外围

器件都已普遍CMOS化。采用CMOS工艺后，单片机具有极佳的低功耗和功耗管理功能。现在新的单片机的功耗越来越低，特别是很多单片机都设置了多种工作方式，包括等待、暂停、睡眠、空闲、节电等工作方式。MCS－51系列的8031单片机推出时的功耗达630mW，而现在的单片机功耗普遍都在100mW左右，有的只有几十微瓦。

MSP430系列单片机是低功耗单片机的典型代表。

2. RISC体系结构的发展

早期单片机大多是复杂指令集(Complex Instruction Set Computer，CISC)结构体系，即所谓的冯·诺伊曼结构，如MCS－51系列单片机。采用CISC结构的单片机数据线和指令线分时复用，其指令丰富，功能较强，但取指令和取数据不能同时进行，速度受限。由于指令复杂，指令代码、周期数不统一，指令运行很难实现流水线操作，大大阻碍了运行速度的提高。传统的MCS－51系列单片机，时钟频率为12 MHz时，单周期指令速度仅1MIPS。虽然单片机对运行速度要求远不如通用计算机系统或数字信号处理器(DSP芯片)对运行速度的要求高，但速度的提高仍会带来许多好处，能拓宽单片机的应用领域。

采用精简指令集(Reduced Instruction Set Computer，RISC)体系结构的单片机，数据线和指令线分离，即所谓的哈佛结构，这使得取指令和取数据可以同时进行，其指令较同类CISC单片机指令包含更多的处理信息，执行效率更高，速度也更快。

Microchip公司的PIC系列、Atmel公司的AT90S系列、SAMSUNG公司的KS57C系列、义隆公司的EM－78系列等多采用RISC结构。

3. ISP及基于ISP的开发应用

目前，片内带E^2PROM的单片机的广泛使用，推动了“在系统可编程”(ISP)技术的发展。在ISP技术基础上，首先实现了目标程序的串行下载，从而促使了模拟仿真开发方式的兴起。在单时钟、单指令运行的RISC结构单片机中，可实现PC通过串行电缆对目标系统的仿真调试。上述仿真技术，使远程调试(即对原有系统方便地更新软件、修改软件和对软件进行远程诊断)成为现实。

现在很多单片机的程序存储器和数据存储器都采用Flash存储器件，可以在线电擦写，并且断电后数据不丢失，系统开发阶段使用十分方便，在小批量应用系统中得到了广泛应用。

1.3　典型单片机简介

1.3.1　MCS－51系列单片机

MCS－51是Intel公司生产的8051单片机系列名称。

MCS－51系列单片机以其良好的开放式结构、种类众多的支持芯片、丰富的软件资源，在我国应用十分广泛。其技术特点是完善了外部总线，确立了单片机的控制功能。外部并行总线规范化为16位地址总线，以寻址外部64 KB程序存储器和数据存储器空间，8位数据总线和相应的控制总线，形成完整的并行三总线结构。

MCS－51系列单片机采用两种生产工艺：一是HMOS工艺(高密度短沟道MOS工艺)；二是CHMOS工艺(互补金属氧化物的HMOS工艺)。CHMOS是CMOS和HMOS

的结合，既保持了 HMOS 高速度和高密度的特点，还具有 CMOS 的低功耗特点。在产品型号中凡带有字母“C”的即为 CHMOS 芯片(如 80C51 等)，CHMOS 芯片的电平既与 TTL 电平兼容，又与 CMOS 电平兼容。

8031 是最早、最基本的产品，该系列的其他单片机都是在 8031 的基础上通过增加功能而来的。

80C51 是 MCS-51 系列中 CHMOS 工艺的一个典型品种。**其他厂商以 8051 为基核开发的基于 CMOS 工艺的单片机产品统称为 80C51 系列，而 MCS-51 系列和 80C51 系列统称为 51 系列单片机**(本书在后面的章节中一般会用 MCS-51 单片机来表述)。当前常用的 51 系列单片机主要产品有 Intel 公司的 80C31、80C51、87C51、80C32、80C52、87C52 等，Atmel 公司的 AT89C51、AT89C52、AT89C2051、AT89S51 等。另外，还有 Philips、华邦、Dallas、Siemens(Infineon)等公司的许多产品，在此不一一列举。

51 系列单片机分类及性能指标见表 1-1。

表 1-1 51 系列单片机分类及性能指标

分类		芯片型号	存储器类型及字节数		片内其他功能单元数量			
			ROM	RAM	并行口	串行口	定时器/计数器	中断源
总线型	基本型	80C31	无	128B	4 个	1 个	2 个	5 个
		80C51	4KB 掩模	128B	4 个	1 个	2 个	5 个
		87C51	4KB EPROM	128B	4 个	1 个	2 个	5 个
		89C51/89S51	4KB Flash ROM	128B	4 个	1 个	2 个	5 个
	增强型	80C32	无	256B	4 个	1 个	3 个	6 个
		80C52	8KB 掩模	256B	4 个	1 个	3 个	6 个
		87C52	8KB EPROM	256B	4 个	1 个	3 个	6 个
		89C52/89S52	8KB Flash ROM	256B	4 个	1 个	3 个	6 个
非总线型		89C2051	2KB Flash ROM	128B	2 个	1 个	2 个	5 个
		89C4051	4KB Flash ROM	128B	2 个	1 个	2 个	5 个

1.3.2 AT89 系列单片机

AT89 系列单片机是 Atmel 公司的 8 位 Flash 单片机系列。这个系列单片机的最大特点是在片内含有 Flash 存储器，开发十分便捷，是 80C51 系列的主流单片机。AT89 系列单片机是以 8051 核为基础构成的，所以，它和 MCS-51 系列单片机是完全兼容的，可以替代以 MCS-51 为基础的单片机系统。对于熟悉 8051 的用户来说，用 Atmel 公司的 89 系列的 AT89C51(或 AT89S51)取代 8051 的系统设计，是轻而易举的事。本书许多案例中的单片机就是以 AT89C51 为例的(但我们在书中还是统一称为 MCS-51 单片机)。

AT89 系列单片机的主要型号有 AT89C51、AT89C52、AT89C2051、AT89S51、AT89S52 等。

89S51 是 89C51 的升级版本，89SXX 可以向下兼容 89CXX 等 51 系列芯片。89S51 有

ISP 在线编程功能；最高工作频率为 33 MHz；内部集成看门狗计时器；带有全新的加密算法，程序的保密性大大加强；电源范围宽达 4～5.5 V。

AT89 系列单片机具有以下优点：

(1) 内部含 Flash ROM。在系统的开发过程中，可以十分容易地进行程序修改，这大大缩短了系统的开发周期，同时在系统工作过程中能有效地保存一些数据信息，即使外部电源损坏也不会影响到信息的保存。

(2) 和 MCS-51 系列单片机引脚兼容。由于 AT89 系列单片机的引脚是和 MCS-51 系列单片机的引脚完全一样的，所以可以用 AT89 系列单片机替代 MCS-51 系列单片机，这时不管采用 40 引脚或是 44 引脚的芯片，只要用相同封装的芯片直接取代即可。

(3) 静态时钟方式。AT89 系列单片机采用静态时钟方式，可以节省电能，这对于降低便携式产品的功耗十分有用。

1.3.3　PIC 系列单片机

PIC(Peripheral Interface Controller)系列单片机是一种用来控制外围设备的可编程集成电路，是由美国 Microchip 公司推出的单片机系列产品。PIC 系列单片机采用了 RISC 结构，其高速度、低电压、低功耗、大电流 LCD 驱动能力和低价位 OTP(一次性编程)技术等都体现出单片机产业的新趋势。PIC 系列单片机在电脑外设、家电、通信设备、智能仪器、汽车电子等各个领域得到了广泛应用，现今的 PIC 系列单片机已经是世界上最有影响力的嵌入式微控制器之一，如 PIC10XX、PIC16XX、PIC24XX、dsPIC30XX、PIC32XX 等。

PIC 系列单片机具有以下优点：

(1) 适用性广。PIC 系列单片机最大的特点是从实际出发，重视产品的性能与价格比，靠发展多种型号来满足不同层次的应用要求。PIC 系列单片机从低到高有几十个型号，可以满足各种需要。其中，PIC12C508 单片机仅有 8 个引脚，是世界上最小的单片机。

(2) 运行效率高。PIC 系列单片机的精简指令集(RISC)使其执行效率大为提高。PIC 系列 8 位 CMOS 单片机具有独特的 RISC 结构，使指令具有单字长的特性，且允许指令码的位数可多于 8 位的数据位数。这与传统的采用 CISC 结构的 8 位单片机相比，可以达到 2∶1 的代码压缩，速度提高 4 倍。

(3) 开发环境优越。单片机开发系统的实时性是一个重要指标。MCS-51 系列单片机的开发系统大都采用高档型号仿真低档型号，实时性不尽理想。PIC 单片机在推出一款新型号的同时推出相应的仿真芯片，所有的开发系统由专用的仿真芯片支持，实时性非常好。

(4) 可靠性高。PIC 系列单片机的引脚具有防瞬态能力，通过限流电阻可以接至 220V 交流电源，可直接与继电器控制电路相连，无需光电耦合器隔离，给应用带来极大方便。PIC 系列单片机自带看门狗定时器，可以用来提高程序运行的可靠性。

(5) 保密性好。PIC 系列单片机以保密熔丝来保护代码，用户在烧入代码后熔断熔丝，别人再也无法读出，除非恢复熔丝。目前，PIC 系列单片机采用熔丝深埋工艺，恢复熔丝的可能性极小。

1.3.4 MSP430 系列单片机

MSP430 系列单片机是美国德州仪器公司(TI)1996 年开始推向市场的一种 16 位超低功耗、具有精简指令集(RISC)的混合信号处理器(Mixed Signal Processor)。之所以称为混合信号处理器，是由于其针对实际应用需求，将多个不同功能的模拟电路、数字电路模块和微处理器集成在一个芯片上，以提供"单片"解决方案。该系列单片机多应用于需要电池供电的便携式装置中。MSP430 系列单片机具有以下优点：

(1) 处理能力强。MSP430 系列单片机是一个 16 位的单片机，采用了精简指令集(RISC)，具有丰富的寻址方式(7 种源操作数寻址、4 种目的操作数寻址)、简洁的 27 条内核指令以及大量的模拟指令；寄存器以及片内数据存储器都可参与多种运算；还有高效的查表处理指令。这些特点保证了可编制出高效率的源程序。

(2) 运算速度快。MSP430 系列单片机能在 25 MHz 晶振的驱动下实现 40 ns 的指令周期。16 位的数据宽度、40 ns 的指令周期以及多功能的硬件乘法器(能实现乘法运算)相配合，能实现数字信号处理的某些算法(如 FFT 等)。

(3) 超低功耗。MSP430 系列单片机的电源电压采用的是 1.8～3.6 V 电压，使芯片整体上处于较低功耗运行状态。独特的时钟系统设计，在 MSP430 系列中有不同的时钟系统：基本时钟系统、锁频环时钟系统和 DCO 数字振荡器时钟系统。可以只使用一个晶体振荡器，也可以使用两个晶体振荡器。由时钟系统产生 CPU 和各功能所需的时钟。并且这些时钟可以在指令的控制下打开和关闭，从而实现对总体功耗的控制。在实时时钟模式下，电流可低到 0.3～2.5 μA；而在 RAM 保持模式下，电流最低可达 0.1 μA。

(4) 片内资源丰富。MSP430 系列单片机都集成了较丰富的片内外设。它们分别是看门狗(WDT)、模拟比较器 A、定时器 A0、定时器 A1、定时器 B0、UART、SPI、I^2C、硬件乘法器、液晶驱动器、10 位/12 位 ADC、16 位 Σ-Δ ADC、DMA、I/O 端口、基本定时器(Basic Timer)、实时时钟(RTC)和 USB 控制器等若干外围模块的不同组合。这些片内外设为系统的单片解决方案提供了极大的便利。

(5) 方便高效的开发环境。MSP430 系列有 OTP 型、Flash 型和 ROM 型三种类型的器件，这些器件的开发手段不同。OTP 型和 ROM 型的器件使用仿真器开发，开发成功之后烧写或掩模芯片；Flash 型的器件则有十分方便的开发调试环境，因为器件片内有 JTAG 调试接口，还有可电擦写的 Flash 存储器，因此采用先下载程序到 Flash 存储器内，再在器件内通过软件控制程序的运行，由 JTAG 接口读取片内信息供开发者调试使用。这种方式只需要一台 PC 和一个 JTAG 调试器，而不需要仿真器和编程器。

1.4 单片机的应用

单片机技术的发展速度十分惊人。时至今日，单片机技术已经发展得相当成熟，成为计算机技术的一个独特而又重要的分支。单片机的应用领域也日益广泛，特别是在工业控制、仪器仪表、汽车电子、家用电器等领域的智能化方面，扮演着极其重要的角色。

1.4.1　单片机的应用特点

单片机的特点很多，这里仅从应用的角度讨论单片机以下几个方面的特点。

1. 控制系统在线应用

控制系统在线应用中，由于单片机与控制对象联系密切，所以不但对单片机的性能要求高，而且对开发者的要求也很高，他们既要熟练掌握单片机，还要了解控制对象，懂得传感技术，具有一定的控制理论知识等。

2. 软/硬件结合

虽然单片机的引入使控制系统大大“软化”，但与其他计算机应用系统相比，单片机控制应用中的硬件内容仍然较多，所以说单片机控制应用具有软/硬件相结合的特点。为此，在单片机的应用设计中需要软、硬件统筹考虑，开发者不但要熟练掌握软件编程技术，而且还要具备较扎实的单片机外围硬件电路设计方面的理论和实践知识。

3. 应用现场环境恶劣

通常，单片机应用现场的环境比较恶劣，电磁干扰、电源波动、冲击震动、高低温等因素都会影响系统工作的稳定性。此外，无人值守环境也对单片机系统的稳定性和可靠性提出了更高的要求。所以，稳定性和可靠性在单片机应用系统中具有十分重要的意义。

1.4.2　单片机的应用领域

提到单片机的应用，有人会这样说：“凡是能想到的地方，单片机都可以用得上。”这并不夸张。由于全世界单片机的年产量以亿计，应用范围之广，花样之多，一时难以详述，下面仅列举一些典型的应用领域或场合。

1. 智能仪器仪表

单片机用于各种仪器仪表，既提高了仪器仪表的使用功能和精度，也使得仪器仪表更加智能化，同时还简化了仪器仪表的硬件结构，从而可以方便地完成仪器仪表产品的升级换代。典型的智能仪器仪表如各种智能测量仪表、智能传感器等。

2. 机电一体化产品

机电一体化产品是集机械技术、微电子技术、自动化技术和计算机技术于一体，具有智能化特征的各种机电产品。单片机在机电一体化产品的开发中可以发挥巨大的作用。典型产品如机器人、数控机床、自动包装机、医疗设备等。

3. 实时工业控制

单片机还可以用于各种物理量的采集与控制。电流、电压、温度、液位、流量等物理参数的采集和控制均可以利用单片机方便地实现。在这类系统中，利用单片机作为系统控制器，可以根据被控对象的不同特征采用不同的智能算法，实现期望的控制目标，从而提高生产效率和产品质量。典型应用如电机转速控制、温度控制、自动生产线等。

4. 分布式系统的前端模块

在较复杂的工业系统中，经常要采用分布式测控系统完成大量的分布参数的采集。在这类系统中，采用单片机作为分布式系统的前端采集模块，具有运行可靠、数据采集方便灵活、成本低廉等一系列优点。

5. 家用电器

家用电器是单片机的又一重要应用领域，前景十分广阔。典型应用如空调器、电冰箱、洗衣机、电饭煲等。这类应用常常采用专用单片机，以达到降低成本的目的。

另外，在电信设备、计算机外围设备、办公自动化设备、汽车、军用装备等领域均有单片机的广泛应用，如汽车自动驾驶系统、航天测控系统、通信系统等。

思考与练习

1. 什么是单片机？单片机由哪些基本部件组成？
2. 为什么说单片机是典型的嵌入式系统？
3. 单片机有什么特点？
4. 什么是单片机应用系统？什么是单片机开发系统？二者之间有何关系？
5. 单片机的主要发展方向是什么？主要应用领域有哪些？
6. MCS－51 系列单片机如何进行分类？各类特点如何？
7. AT89 系列单片机有什么优点？

第 2 章　单片机基本结构和工作原理

本章主要介绍 MCS－51 单片机的部件及内部结构、外部引脚、CPU 工作时序、工作方式等，重点讲述单片机的存储器结构、I/O 电路、辅助电路等内容。

2.1　单片机的组成和内部结构

从结构上看，MCS－51 单片机与通用微型计算机没有什么区别，都是由 CPU 加上一些功能部件组成的。只是单片机将这些部件都集成到了一个芯片上，使用时只需再添加一些外围器件就可以构成单片机应用系统。

2.1.1　单片机的组成

单片机是在一块芯片中集成了 CPU、RAM、ROM、定时器/计数器和 I/O 端口等多种基本功能部件，如图 2－1 所示。单片机有基本型和增强型两种，基本型的代表产品为 8051，增强型的代表产品为 8052，两者的主要区别在于内部存储器的大小和定时器/计数器的个数不同。

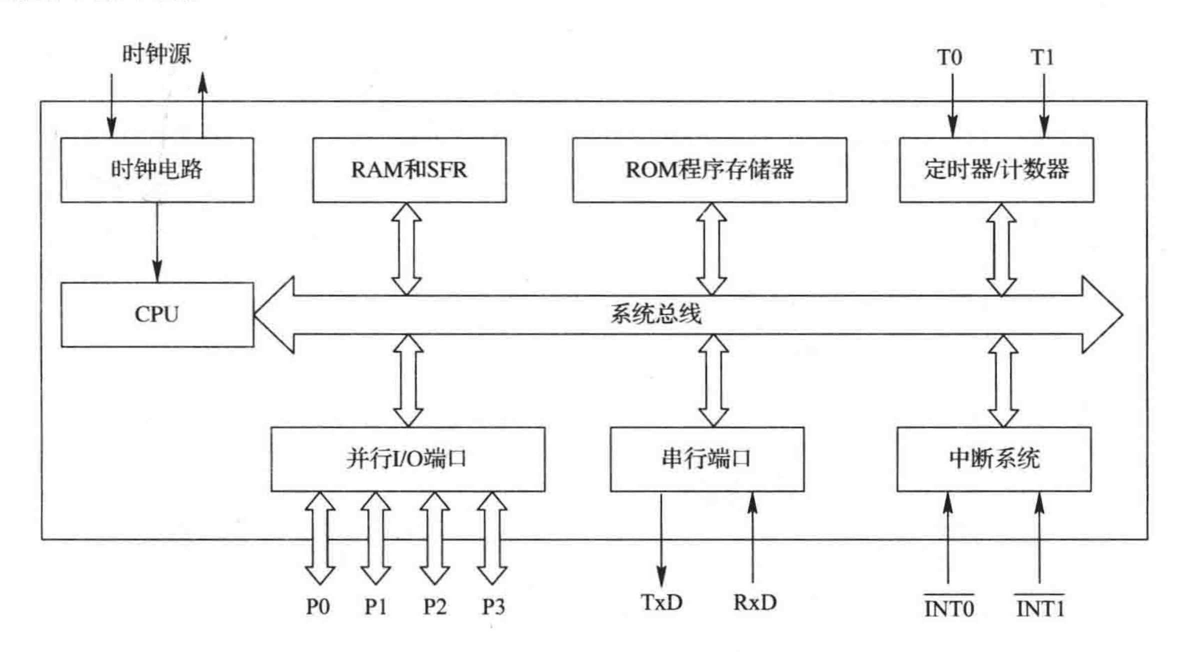


图 2－1　MCS－51 单片机的功能框图

单片机内部通常包含下列一些部件：

- 一个 8 位 CPU；
- 一个片内振荡器及时钟电路；

- 4 KB ROM 程序存储器(8031 没有片内 ROM，增强型为 8 KB)；
- 128 B RAM 数据存储器(增强型为 256B)；
- 两个 16 位定时器/计数器(增强型为三个)；
- 可寻址 64 KB 外部数据存储器和 64KB 外部程序存储器空间的控制电路；
- 32 条可编程的 I/O 口(四个 8 位并行 I/O 端口)；
- 一个可编程全双工串行口；
- 具有五个中断源、两个优先级嵌套中断结构(增强型为六个中断源)。

2.1.2 单片机的内部逻辑结构

单片机的各功能部件通过内部总线连接在一起，其中包括算术逻辑单元(ALU)、累加器(ACC 或 A)、ROM、RAM、指令寄存器(IR)、程序计数器(PC)、定时器/计数器、I/O 接口电路、程序状态字寄存器(PSW)、堆栈指针(SP)、数据指针(DPTR)等，详细的内部结构如图 2-2 所示。

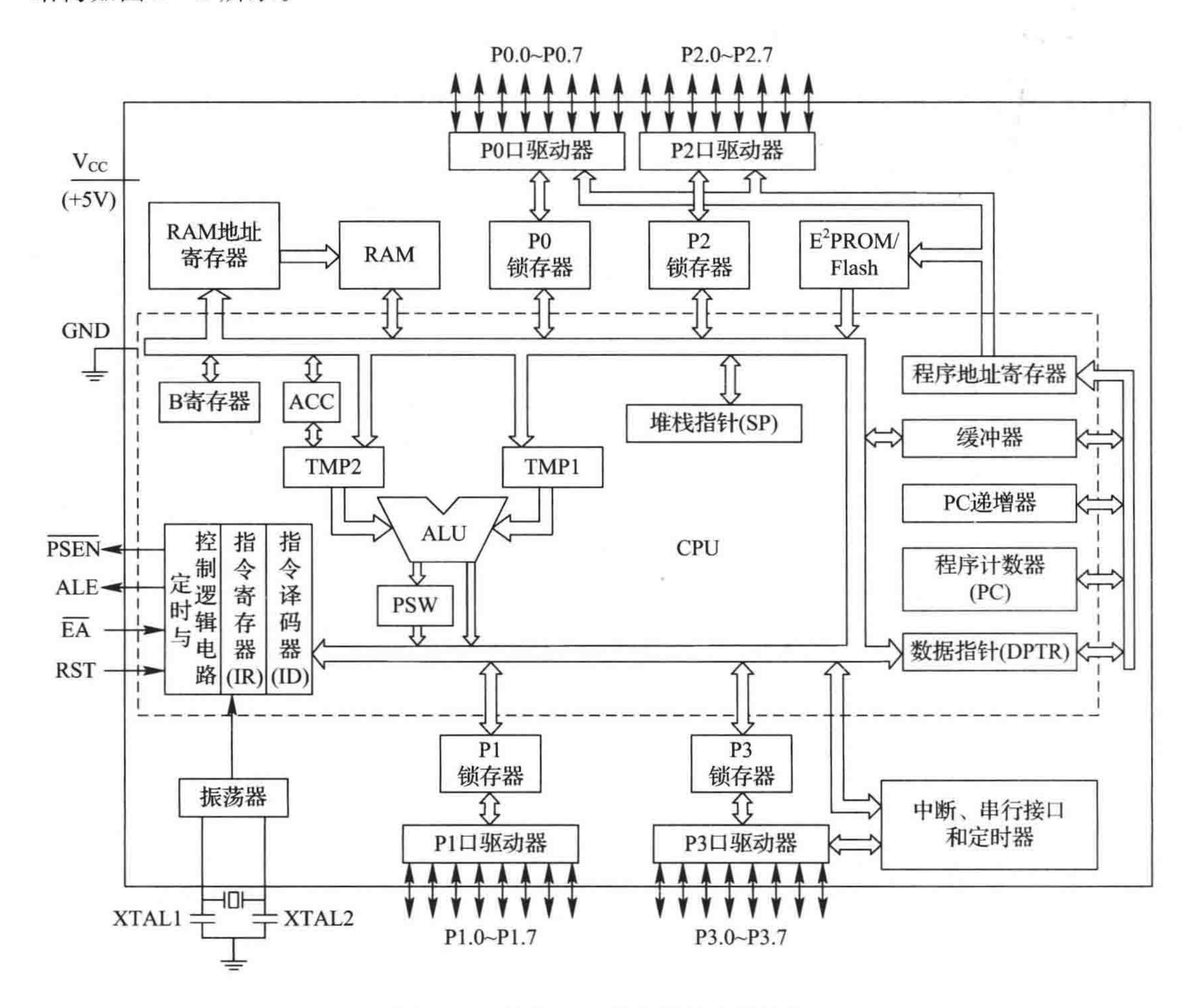


图 2-2 MCS-51 单片机的内部结构

2.1.3　CPU 的内部结构

单片机内部核心部分是一个 8 位高性能微处理器 CPU，由运算器和控制器等部件组成（即图 2－2 中虚线框内部分），它是单片机的头脑和心脏，用以完成各种运算和控制操作。

1. 运算器

运算器的主要功能是进行算术运算和逻辑运算、位运算、数据中转与处理，并将操作结果的状态信息送至程序状态字寄存器中。运算器主要包括算术逻辑单元、累加器、B 寄存器、暂存寄存器、程序状态字寄存器等。

（1）算术逻辑单元（ALU）。算术逻辑单元是运算器的核心部件，实质上是全加器，可以用于对数据进行加、减、乘、除等算术运算，还能对数据进行与、或、异或、循环、置 1、清 0 等逻辑运算，并具有数据传送、程序转移等功能。ALU 不能由程序读写。

（2）累加器（ACC 或 A）。累加器是一个 8 位寄存器。很多运算都要通过累加器提供操作数，多数运算结果也在 ACC 中存放。

（3）B 寄存器。B 寄存器是为乘法和除法而设置的。在进行乘法和除法运算时，A 和 B 组成寄存器对，记为 AB。在不执行乘法和除法运算时，B 寄存器可以作为一个普通寄存器使用。

（4）暂存寄存器（TMP1 和 TMP2）。暂存寄存器用于暂时存储数据总线或其他寄存器送来的操作数，作为 ALU 的数据源，向 ALU 提供操作数。暂存寄存器不能由程序读写。

（5）程序状态字寄存器（PSW）。程序状态字寄存器是一个 8 位的专用寄存器，主要用于存放当前运算结果的状态。

2. 控制器

控制器的主要功能是识别指令，并根据指令的性质控制单片机内部的各个功能部件，使其协调工作。单片机执行指令严格受控制器的控制，它们从程序存储器中读取指令，送入寄存器，然后进行译码，译码的结果与时序电路结合，发出操作信号。程序的执行就是不断重复这一过程。

控制器包含程序计数器、指令寄存器、指令译码器、数据指针、堆栈指针、定时与控制逻辑电路等。

（1）程序计数器（PC）。程序计数器是一个 16 位专用寄存器，用于存放将要执行指令的地址，具有自动加 1 功能。PC 没有对应的寄存器，程序无法直接设置其中的数据。当 CPU 取指时，PC 的内容首先送至内部地址总线上，然后从程序存储器中取出指令，PC 内容自动加 1，以保证程序的顺序执行。

在执行转移、子程序调用指令和中断响应时，PC 的内容不再加 1，而是由指令或中断响应过程自动给 PC 置入新的地址。单片机复位时，PC 自动清 0，即装入 0000H（H 表示前面的数字为十六进制），从而保证复位后程序从 0000H 开始执行。

（2）指令寄存器（IR）。指令寄存器是一个 8 位寄存器，用于寄存等待执行的指令。IR 不能由程序读写。

（3）指令译码器（ID）。指令译码器对指令寄存器中的指令进行译码，产生执行该指令所需的一系列控制时序信号，以执行相应的操作。ID 不能由程序读写。

(4) 数据指针(DPTR)。数据指针(DPTR)是一个16位专用寄存器，通常在访问外部数据存储器时作地址指针使用。

(5) 堆栈指针(SP)。堆栈指针(SP)是一个8位专用寄存器，用于存放堆栈栈顶的地址。

(6) 定时与控制逻辑电路。定时与控制逻辑电路是控制器的核心部件之一，它的任务是产生各种控制信号，协调各功能部件的工作。单片机内部设有振荡电路，只需外接石英晶体(晶振)和微调电容就可产生振荡脉冲信号，经过二分频后，生成时钟信号，是单片机的基本节拍，单片机就是在这个节拍的控制下协调工作的。

2.1.4 单片机的其他结构模块

1. 内部RAM

MCS-51单片机的内部RAM用于存放单片机运行时的数据，其寻址范围为256字节，其中低128字节可以作为内部随机访问存储器，高128字节被特殊功能寄存器占用。

2. 内部ROM

MCS-51单片机中的ROM主要用来存放程序，也可以存放一些常数和表格。单片机运行时，ROM中的内容是不能修改的。MCS-51系列单片机可分为内部无ROM型(如8031)和内部有ROM型(如8051)两种。在多数情况下，无论是8031还是8051，都必须根据实际需要外接EPROM型程序存储器。而对于后来出现的，内部含有E^2PROM或Flash ROM类型程序存储器的AT89CXX/AT89SXX系列单片机(如市面上最常见的AT89S51和AT89C51，内置4 KB的Flash ROM)，通常不需要外部扩展程序存储器。

3. 定时器/计数器

MCS-51单片机有两个16位定时器/计数器，能够实现精确定时和对外部脉冲信号计数，可以用于定时控制、延时，以及对外部事件进行计数和检测等。

4. 中断

MCS-51单片机有五个中断源，即两个外部中断、两个定时/计数中断和一个串行通信中断，同时有两个中断优先级。

5. 串行通信口

MCS-51单片机有一个采用通用异步工作方式的全双工串行通信口(串行口)，可以同时接收和发送数据。

6. I/O口

MCS-51单片机有四个8位的并行端口，分别为P0、P1、P2和P3，这些端口可以用于输入或输出，除了P1外，每个I/O口还有第二功能(详见2.4节)。

7. 内部总线

如图2-2所示，单片机所有功能模块都是通过内部总线连接起来的，从而构成一个完整的单片计算机系统。单片机内部的地址信号、数据信号和控制信号都是通过内部总线传送的。

2.2 单片机的外部引脚及功能

MCS-51系列单片机根据不同的型号，其引脚数目和封装形式也有很大差别。常见

的有 40 引脚的双列直插(DIP)封装形式和 44 引脚的 PLCC 封装形式，新型的单片机还有 44 引脚的 TQFP 封装形式。图 2-3 所示是两种常见封装形式的引脚配置图。

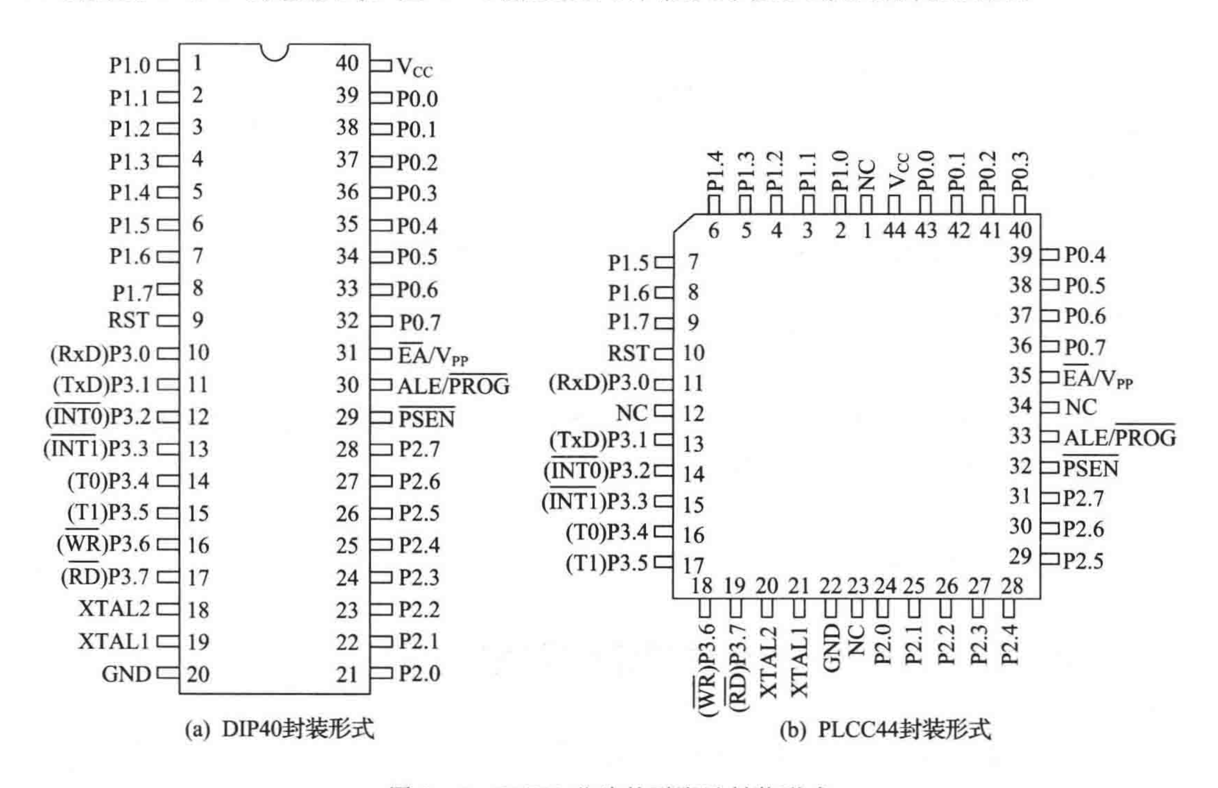


图 2-3 89C51 芯片的引脚及封装形式

下面以 MCS-51 系列单片机中典型芯片 89C51 为例介绍其引脚排列和定义。89C51 的引脚按功能分类主要有 I/O 引脚、控制引脚和电源与晶振引脚。

2.2.1 I/O 引脚

I/O 引脚即输入/输出端口，在某一时刻只能作为输入口或输出口使用，所以是准双向口，MCS-51 单片机有 P0(P0.0～P0.7)、P1(P1.0～P1.7)、P2(P2.0～P2.7)、P3(P3.0～P3.7)四个 8 位准双向输入/输出端口，每个端口都有锁存器、输出驱动器和输入缓冲器。四个 I/O 端口都可作为输入/输出口使用，其中 P0、P2 和 P3 口还可以组成三总线，用于外围芯片扩展(详见第 7 章)。

2.2.2 控制引脚

RST：复位输入端，高电平有效。当振荡器运行时，在 RST 引脚上出现两个机器周期以上的高电平使单片机复位。

ALE/$\overline{\text{PROG}}$：当访问外部存储器时，ALE(地址锁存使能)的输出用于锁存地址的低位字节；当不访问外部存储器时，ALE 端以固定频率(为振荡频率的 1/6)输出脉冲信号，所以 ALE 可以用作其他器件的时钟源。需要注意的是，每当访问外部数据存储器时，将跳过一个 ALE 脉冲。在对片内 ROM 编程(写数据)时，该引脚的第二功能用于输入编程脉冲$\overline{\text{PROG}}$。

$\overline{PSEN}$：外部程序存储器取指令使能端，低电平有效。在访问外部ROM时，该信号自动产生，每个机器周期输出两个脉冲。

$\overline{EA}/V_{PP}$：外部程序访问允许端。当$\overline{EA}$为高电平时，CPU从内部程序存储器执行指令，当PC值超过片内程序存储器最大地址范围时，将自动转向外部存储器执行程序。当$\overline{EA}$为低电平时，CPU只从外部程序存储器执行指令。在片内ROM编程期间，该引脚的第二功能用于加12V的编程允许电源V_{PP}。

2.2.3 电源与晶振引脚

V_{CC}：电源端，通常的电源电压为+5 V。

GND：接地端。

XTAL1：接外部晶振的一个引脚。在单片机内部，它是构成片内振荡器反相放大器的输入端。当采用外部时钟时，该引脚接外部时钟信号。

XTAL2：接外部晶振的另一个引脚。在单片机内部，它是构成片内振荡器反相放大器的输出端。当采用外部时钟时，此引脚应悬空。

2.3 单片机的存储器结构

MCS-51单片机的存储器组织采用哈佛结构，即数据存储器与程序存储器使用不同的逻辑空间、不同的物理存储、不同的寻址方式和不同的访问时序。

从物理上看，MCS-51单片机有4个存储空间：内部程序存储器、外部(片外)程序存储器、内部数据存储器、外部数据存储器。从开发者的角度看，MCS-51单片机有3个存储地址空间：芯片内外统一的程序存储空间、内部数据存储空间和外部数据存储空间。

2.3.1 程序存储器

程序存储器常用来存放程序、表格和常数，也称为ROM。ROM以程序计数器(PC)作为地址指针，通过16位地址总线寻址，可寻址的地址空间为64 KB，地址范围为0000H～FFFFH。MCS-51单片机可分为内部无ROM型(8031)和内部有ROM型(8051)两种。

1. 片内与片外程序存储器的选择

对于无内部ROM型的单片机，必须**使用外部ROM，这时$\overline{EA}$引脚必须接低电平**。对于有内部ROM型的单片机，但不使用内部ROM，如8051内部是掩模ROM，一般不用此ROM，这时$\overline{EA}$引脚也必须接低电平，外部ROM的地址从0000H开始。对于内部含有Flash ROM的AT89C/AT89S系列单片机，可以使用内部ROM。如果要**使用内部ROM，则$\overline{EA}$引脚必须接高电平**，当程序执行超出内部存储空间时，单片机会自动转向外部空间，内部ROM的地址范围是0000H～0FFFH，外部ROM的地址从1000H开始。无论是否使用内部ROM，其程序存储器的地址结构和组织结构是一样的，如图2-4所示。

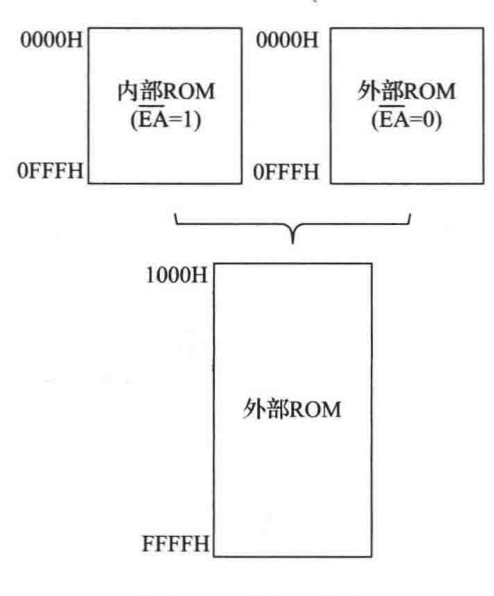


图 2－4　程序存储器

2. 程序存储器低端的特殊单元

在程序存储器中，前面若干个单元地址是中断程序的入口地址，在编写程序时要把这些单元预先进行处理(详见第 4 章)，如表 2－1 所示。其中 0000H 为单片机复位后执行的第一条指令的存放地址。其余为中断向量，即单片机中断服务程序的第一条指令存放地址。

表 2－1　复位及中断入口地址

地址	功能
0000H	复位
0003H	外部中断 0
000BH	定时器/计数器 0
0013H	外部中断 1
001BH	定时器/计数器 1
0023H	串行口中断

3. 程序存储器中的程序代码及其观察

在单片机应用系统开发过程中，程序存储器中程序代码(十六进制)可以在 Keil μVision 集成开发环境的观察窗口中看到(详见 9.2.4 小节)，程序存储器的映射关系及观察界面如图 2－5 所示。

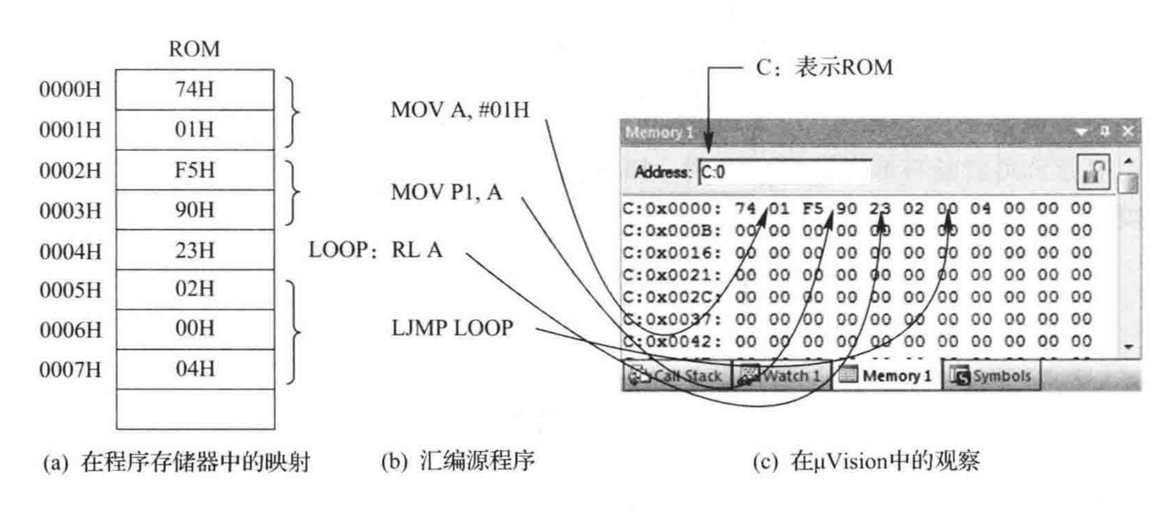


图 2-5　程序存储器映射关系及观察界面

2.3.2　数据存储器

数据存储器常用来存放数据，也称为 RAM。MCS-51 单片机的数据存储器无论在物理上或逻辑上都分为两个地址空间：一个为内部数据存储器(内部 RAM)，如图 2-6 所示，访问内部 RAM 用 MOV 指令，使用 8 位地址，其寻址空间为 256B；另一个为外部数据存储器(外部 RAM)，如图 2-7 所示，访问外部 RAM 用 MOVX 指令，通常用数据指针 DPTR 来寻址，使用 16 位地址，其寻址空间为 64KB。

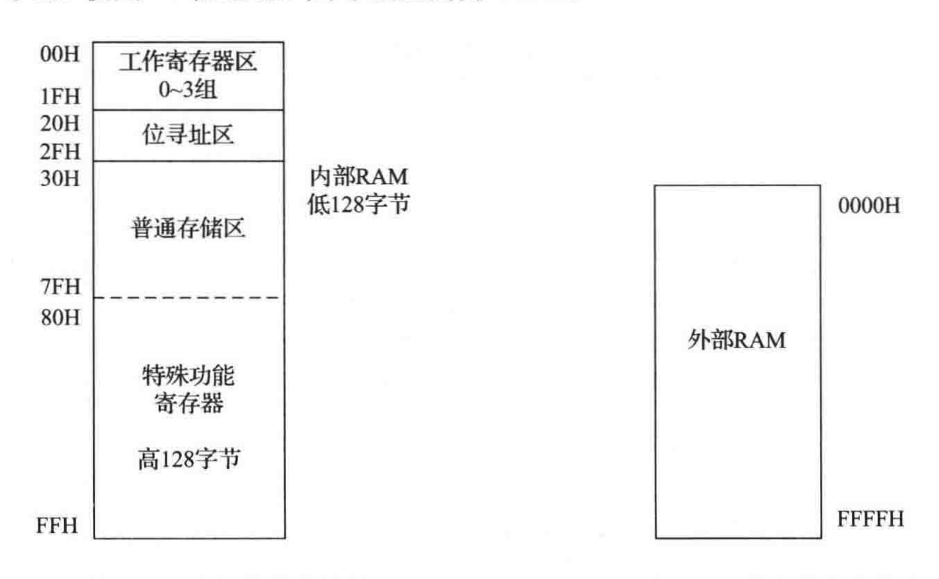


图 2-6　内部数据存储器　　　　图 2-7　外部数据存储器

内部 RAM 有最灵活的地址空间分割，它分成物理上独立而又性质不同的几个区：由 00H～7FH(0～127)单元组成的低 128 字节地址空间的 RAM 区；由 80H～FFH(128～255)单元组成的高 128 字节地址空间的特殊功能寄存器(又称 SFR)区。

内部 RAM 按使用方法分为以下几个部分。

1. 工作寄存器区

单片机对工作寄存器的操作具有指令数量多、程序代码短、执行速度快的特点。在程

序设计时，应尽可能地使用工作寄存器。内部RAM的00H～1FH区域为工作寄存器区。工作寄存器一共有4组，每组又包括8个寄存器，记为R0～R7。4组工作寄存器可根据PSW(程序状态字寄存器，详见2.3.3小节)中的RS1、RS0选择，如表2-2所示。每个时刻，只能使用一组工作寄存器。在程序执行过程中可以通过设置RS1和RS0改变工作寄存器组。要注意，这时的存储空间发生变化，即R0～R7的内容也发生变化。**单片机初始上电时，工作寄存器使用第0组(内部RAM的00H～07H)。**编程时，应尽量使用第0组工作寄存器。

表2-2　RS1、RS0与工作寄存器组的对应关系

RS1	RS0	工作寄存器组	工作寄存器	对应RAM中的地址
0	0	第0组	R0～R7	00H～07H
0	1	第1组	R0～R7	08H～0FH
1	0	第2组	R0～R7	10H～17H
1	1	第3组	R0～R7	18H～1FH

2. 位寻址区

内部RAM的20H～2FH区域为位寻址区，见表2-3，这16个单元中的每一位都有一个位地址，位地址范围为00H～7FH。位寻址区的每一位都可以视作软件触发器，可以由程序直接进行位处理。通常把程序的各种状态标志、位控制变量等设在位寻址区内。同样，位寻址区的RAM单元也可以作为一般的数据存储器按字节使用。

表2-3　位寻址区地址对应表

字节地址	位地址							
	D7	D6	D5	D4	D3	D2	D1	D0
20H	07H	06H	05H	04H	03H	02H	01H	00H
21H	0FH	0EH	0DH	0CH	0BH	0AH	09H	08H
22H	17H	16H	15H	14H	13H	12H	11H	10H
23H	1FH	1EH	1DH	1CH	1BH	1AH	19H	18H
24H	27H	26H	25H	24H	23H	22H	21H	20H
25H	2FH	2EH	2DH	2CH	2BH	2AH	29H	28H
26H	37H	36H	35H	34H	33H	32H	31H	30H
27H	3FH	3EH	3DH	3CH	3BH	3AH	39H	38H
28H	47H	46H	45H	44H	43H	42H	41H	40H
29H	4FH	4EH	4DH	4CH	4BH	4AH	49H	48H
2AH	57H	56H	55H	54H	53H	52H	51H	50H
2BH	5FH	5EH	5DH	5CH	5BH	5AH	59H	58H
2CH	67H	66H	65H	64H	63H	62H	61H	60H
2DH	6FH	6EH	6DH	6CH	6BH	6AH	69H	68H
2EH	77H	76H	75H	74H	73H	72H	71H	70H
2FH	7FH	7EH	7DH	7CH	7BH	7AH	79H	78H

3. 普通存储区

内部 RAM 的 30H～7FH 区域为普通存储区，只能按字节寻址，一般用于存放程序执行过程中的临时数据。

4. 堆栈区

在一个程序中，往往需要设定一个后进先出(或者先进后出)的缓冲区，用以保存某些重要数据和地址，**这种后进先出的缓冲区称为堆栈区**。堆栈区原则上可以设在内部 RAM 的任意区域内，只需注意不要与已使用的 RAM 重叠。栈顶的位置由堆栈指针 SP 确定。

2.3.3 特殊功能寄存器

单片机内的锁存器、定时器、串行口数据缓冲器以及各种控制寄存器和状态寄存器都是以特殊功能寄存器(SFR)形式出现的，它们分布在内部 RAM 的 80H～FFH 地址空间范围内。MCS-51 基本型单片机有 21 个 SFR，表 2-4 给出了 MCS-51 单片机的特殊功能寄存器的名称和地址。

表 2-4 特殊功能寄存器

标志符	名　称	地　址
ACC	累加器	E0H
B	B 寄存器	F0H
PSW	程序状态字寄存器	D0H
SP	堆栈指针	81H
DPH	数据指针(DPTR)高字节	83H
DPL	数据指针(DPTR)低字节	82H
P0	P0 口	80H
P1	P1 口	90H
P2	P2 口	A0H
P3	P3 口	B0H
IE	中断允许控制寄存器	A8H
IP	中断优先级控制寄存器	B8H
TMOD	定时器/计数器工作方式寄存器	89H
TCON	中断请求标志寄存器	88H
TH0	定时器/计数器 0(高字节)	8CH
TL0	定时器/计数器 0(低字节)	8AH
TH1	定时器/计数器 1(高字节)	8DH
TL1	定时器/计数器 1(低字节)	8BH
SCON	串行口控制寄存器	98H
SBUF	串行口收发数据寄存器	99H
PCON	电源控制寄存器	87H

特殊功能寄存器只占用了 128 字节的一小部分，这为单片机增加新功能提供了极大的

余地。这些寄存器除了 DPTR 外，都是 8 位寄存器，而 DPTR 的 DPH 和 DPL 也可以按照 8 位寄存器单独使用。

在特殊功能寄存器中，地址尾数是 0 或 8 的寄存器(比如 ACC、PSW、SCON 等)不仅可以按字节访问，也可以按位访问。特殊功能寄存器 A、B、PSW、SP、DPTR 等在 RAM 中的映射关系如图 2-8 所示。

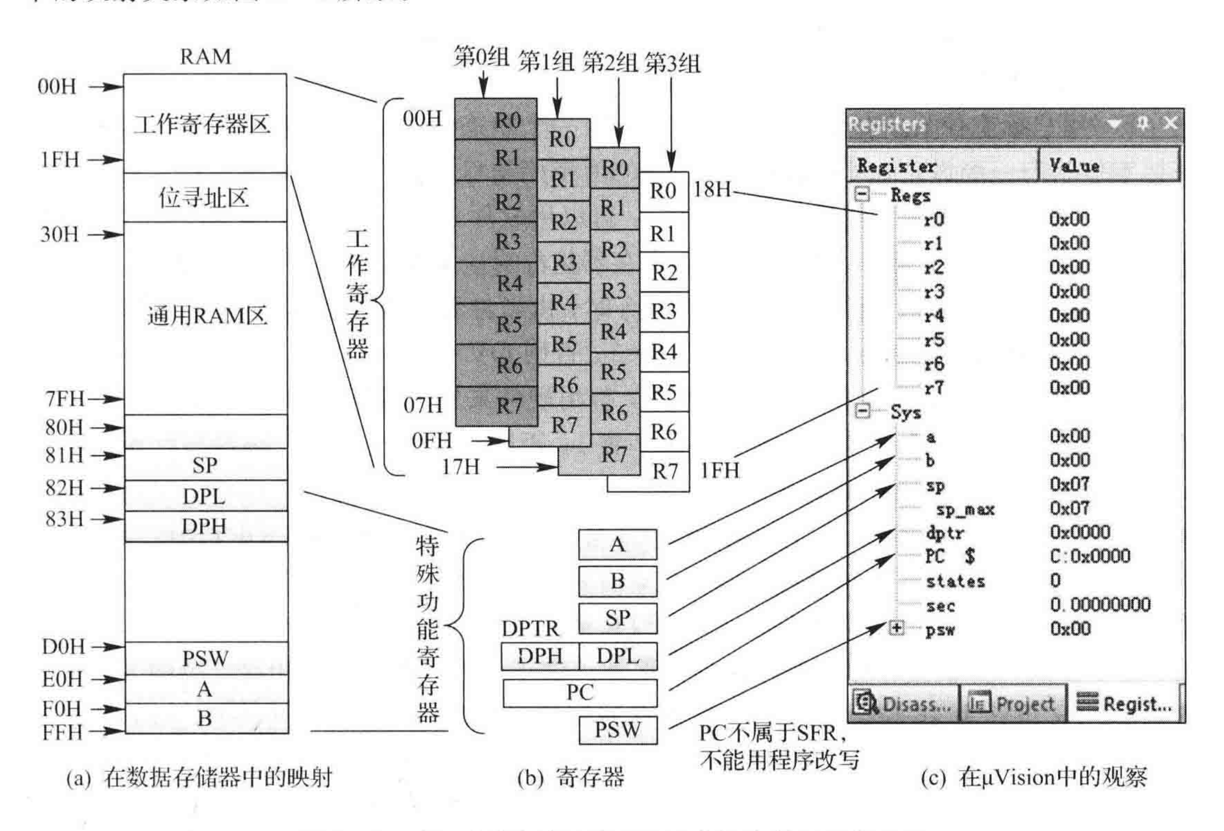


图 2-8　特殊功能寄存器在 RAM 中的映射及观察界面

下面介绍几个常用的特殊功能寄存器。

1. 累加器

累加器是一个 8 位的寄存器，ACC 表示地址(E0H)，寄存器名称为 A。它通过暂存器与 ALU 相连，它是 CPU 工作中使用最频繁的寄存器，用来存放一个操作数或中间结果。累加器在指令中通常用“A”表示，在位操作和堆栈操作指令中则用“ACC”表示。MCS-51 单片机中，只有一个累加器，大部分单操作数指令的操作数取自累加器，许多双操作数指令的一个操作数也取自累加器，在变址寻址方式中累加器被作为变址寄存器使用。

2. 程序状态字寄存器

程序状态字寄存器(PSW)是一个 8 位的专用寄存器，主要用于存放当前运算结果的状态。地址为 D0H，可以按位进行访问，格式如下(第一行是位地址，第二行是位名称，若不能按位操作，则第一行为空格。本书类似内容均采用此种表示方法)：

	D7	D6	D5	D4	D3	D2	D1	D0
(D0H)	D7H	D6H	D5H	D4H	D3H	D2H	D1H	D0H
PSW	Cy	Ac	F0	RS1	RS0	OV	—	P

其中 F0、RS1、RS0 可以用软件设置(即通过编写程序设置),Cy、Ac、OV 和 P 由 CPU 决定(即由单片机内部自动设置)。

Cy(进位标志位):当有进位或借位时,Cy=1;否则,Cy=0。在执行某些算术和逻辑指令时,可以被硬件(指单片机内部的 CPU,表示功能可以自动完成)或软件(表示开发者编写的程序)置 1 或清 0。在布尔处理器中,它被作为位累加器使用。Cy 在程序中一般用 C 表示。

Ac(辅助进位标志):当进行加减运算时,若低 4 位向高 4 位产生进位或借位,则由硬件将其置 1,否则清 0,Ac 被用于 BCD 码调整。

F0(用户标志位):F0 是开发者可以定义的一个状态标记,用软件来使它置 1 或清 0。该标志位状态一经设定,可由软件检测 F0 的值来控制程序执行的方向。

RS1、RS0(工作寄存器组选择控制位):可以用软件来置 1 或清 0,以改变工作寄存器组在 RAM 中的区域。RS1、RS0 与工作寄存器组的对应关系如表 2-2 所示。

OV(溢出标志位):当执行算术运算指令时,由硬件置 1 或清 0,以指示溢出状态。D6 位和 D7 位不同时产生进位或借位时,OV=1,否则 OV=0。

P(奇偶标志位):每次指令执行结束后,都由硬件来置 1 或清 0,以表示累加器 A 中 1 的个数的奇偶性。若 1 的个数为奇数,则 P 置 1,否则 P 清 0。

3. 数据指针

数据指针(DPTR)是一个 16 位的专用地址指针寄存器。编程时,DPTR 既可以作为 16 位寄存器,也可以拆成两个独立的 8 位寄存器,即 DPH(高 8 位字节)和 DPL(低 8 位字节),分别占据 83H 和 82H 两个地址。**DPTR 通常在访问外部数据存储器时作地址指针使用**,用于存放外部数据存储器的存储单元地址。由于外部数据存储器的寻址范围为 64KB,故把 DPTR 设计为 16 位,通过 DPTR 寄存器间接寻址方式可以访问 0000H~FFFFH 全部 64KB 的外部数据存储器空间。

因此,MCS-51 单片机可以外接 64 KB 的数据存储器和扩展 I/O 端口,并可以使用 DPTR 来间接寻址。

4. 堆栈指针

堆栈指针(SP)是一个 8 位寄存器,地址是 81H,用于指示堆栈顶部在内部 RAM 中的位置。可以把 SP 看成一个地址指针,它总是指向堆栈顶端的存储单元。MCS-51 单片机的堆栈是增量式的,即进栈时,SP 的内容是增加的(SP 指针先自动加 1,然后向 SP 指针指向的存储单元送入一个数),出栈时,SP 的内容是减少的。**单片机复位后,SP 初始化默认值为 07H**,使得堆栈事实上由 08H 单元开始。

SP 的内容一经确定,堆栈的位置也就跟着确定了。由于 SP 可以由程序设为不同值,因此堆栈位置是可以浮动的。考虑到 08H~1FH 分属于工作寄存器组的第 1~3 组,若程序设计要用到工作寄存器组的第 1~3 组,则 SP 不能取默认值,而应把堆栈区设置在普通存储区内。

5. I/O 端口的专用寄存器

P0~P3 口寄存器实际上就是 P0~P3(引脚)专用的锁存器,用 P0~P3 表示。MCS-51系列单片机没有专门的端口操作指令,均采用统一的 MOV 指令,直接读写 P0~P3,使用极为方便。

6. 串行数据缓冲器

串行数据缓冲器(SBUF)用于存放待发或已接收到的数据,它实际上由两个独立的寄

存器组成：一个是发送缓冲器，另一个是接收缓冲器。这两个寄存器共享一个地址 99H。

其余的特殊功能寄存器将在以后各章中介绍。

2.4　单片机的 I/O 电路

MCS-51 单片机本身提供了四个 8 位的并行接口，分别记为 P0、P1、P2 和 P3，共有 32 根 I/O 口线。**P0～P3 都是准双向端口，每一根 I/O 口线都能独立地用作输入或输出(可以按位访问)**。四个并行端口结构不同，功能各异，除了可以作为 I/O 口外，P0、P2 和 P3 口还负责提供数据总线、地址总线和控制总线。四个 8 位并行 I/O 端口在汇编指令中也用 P0、P1、P2 和 P3 表示。

I/O 口作为输入口之前，应先向端口写 1，以保证读入正确的输入状态。单片机初始上电时，所有的 I/O 口均处于高电平，这时 I/O 口直接作为输入口，则不需要向端口写 1。

2.4.1　P0 口

1. P0 口的结构

P0 口是一个多功能的 8 位双向并行接口。P0 口某位的内部电路结构如图 2-9 所示，它包含一个输出锁存器(D 触发器)、两个三态缓冲器(三态门 1 和三态门 2)、一个输出驱动电路和一个输出控制电路。

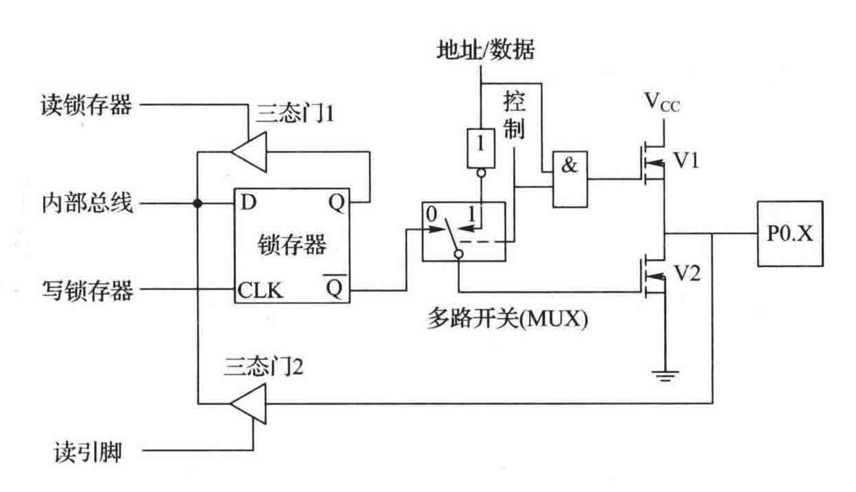


图 2-9　P0 口的内部电路结构

2. P0 口的功能

MCS-51 单片机的 P0 口有两种功能：通用 I/O 接口或地址/数据分时复用总线。

1) 通用 I/O 接口

输出：CPU 内部发出控制电平 0 封锁与门，使与门输出为 0，上方的场效应管 V1 处于截止状态，因此输出驱动级是漏极开路的开漏电路。这样，当写脉冲加在 D 触发器的 CLK 端时，与内部数据总线相连的 D 端数据取反后出现在 $\overline{Q}$ 端，再经下方的场效应管 V2 反相，在 P0 引脚上出现的数据就正好是内部总线的数据。但要注意，由于 P0 口输出驱动电路工作于开漏状态，因此**P0 作为输出口时需要外接上拉电阻**。

输入：P0 口作为 I/O 口使用时的另一种情况是数据由引脚输入，这时使用下方的三态

输入缓冲器(三态门 2)直接读端口引脚处的数据。以上操作称为“读信号”操作。

“读—改—写”操作：端口处于输出状态，将端口当前的数据读入 CPU，在 CPU 中进行运算、修改后，再写到端口输出。需要指出的是，“读—改—写”操作中的“读”是指“读锁存器”，而非“读端口引脚”，因为直接读端口引脚有可能得到错误的读入结果。例如，当用一根端口线去驱动一个晶体管的基极时，如果已经向此端口线写 1，便会令晶体管导通，把引脚拉成低电平。因此，直接读引脚必然会将原先输出的 1 误读为 0，当改为读锁存器后，就能将原先输出的 1 正确读入。图 2-9 上方的三态输入缓冲器(三态门 1)就是为读锁存器 Q 端的数据而设置的。

2) 地址/数据分时复用总线

在单片机应用系统中，P0 口作为地址/数据总线使用分为两种情况。

一种是以 P0 引脚输出地址/数据信息。这时，CPU 内部发出控制信号 1，打开与门，使得多路开关将内部地址/数据线经反相器与场效应管的栅极接通。若地址/数据信号为 0，则该 0 信号一方面经与门使上方的场效应管 V1 截止，另一方面经反相器使下方的场效应管 V2 导通，从而使引脚输出 0 信号；反之，若地址/数据信号为 1，则上方的场效应管 V1 导通，下方的场效应管 V2 截止，引脚输出 1 信号。显然在上述情况中，不必外接上拉电阻。

另一种情况是 P0 口由与其连接的外部存储器输入数据。为了确保数据的正确输入，CPU 在访问外部存储器期间，会在读入数据之前自动地向 P0 口的锁存器写入 FFH。因此，对于用户而言，当 P0 口作为地址/数据总线使用时，它是一个真正的双向口。

3. 负载能力

P0 口输出时能驱动 8 个 LSTTL 负载，即输出电流不小于 800 μA。

2.4.2 P1 口

1. P1 口的结构

图 2-10 所示是 P1 口其中 1 位的结构原理图，P1 口由 8 个这样的电路组成。图中的锁存器起输出锁存作用。场效应管与上拉电阻组成输出驱动器，以增大负载能力。三态门 2 是输入缓冲器，三态门 1 在端口操作时使用。

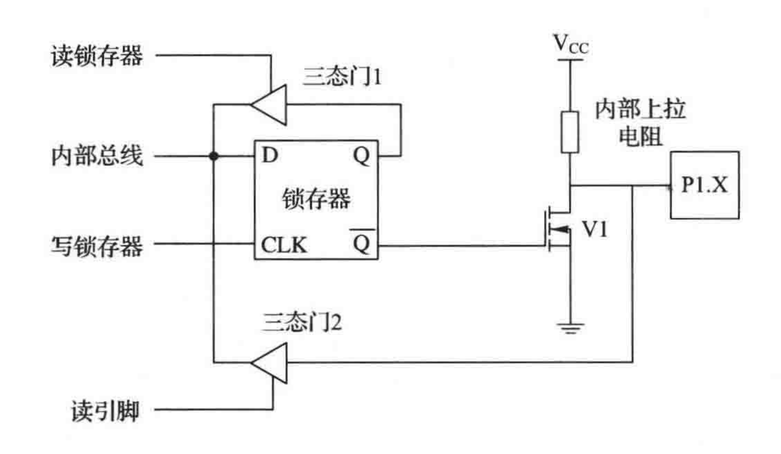


图 2-10 P1 口的内部电路结构

2. P1 口的功能

MCS-51 单片机的 P1 口只有一种功能——通用 I/O 接口。

P1口工作于输出方式：此时数据data经内部总线送入锁存器锁存。如果某位的数据为1，则该位锁存器输出端Q=1，$\overline{Q}$=0，使V1截止，从而在引脚P1.X上出现高电平；反之，如果数据为0，则$\overline{Q}$=1，Q=0，使V1导通，P1.X上出现低电平。

P1口工作于输入方式：控制器发出的读信号打开三态门2，引脚P1.X上的数据经三态门2进入芯片的内部总线。在执行输入操作时，如果锁存器原来寄存的数据Q=0，那么由于$\overline{Q}$=1，将使V1导通，引脚被始终钳位在低电平上，不可能输入高电平。为此，用作输入前，必须先用输出指令置Q=1，使V1截止。单片机复位后，P1口线的状态都是高电平，可以直接用作输入。

MCS-51单片机有不少指令可直接进行端口操作，例如：

```
ANL    P1, #data    ; (P1)←(P1) ∧ data
ORL    P1, #data    ; (P1)←(P1) ∨ data
XRL    P1, A        ; (P1)←(P1)⊕(A)
INC    P1           ; (P1)←(P1)+1
```

3. 负载能力

P1口输出时能驱动4个LSTTL负载，即输出电流不小于400 μA。

2.4.3　P2口

1. P2口的结构

图2-11所示是P2口其中1位的结构原理图，P2口由8个这样的电路组成。P2口的位结构比P1口多了一个转换控制部分。

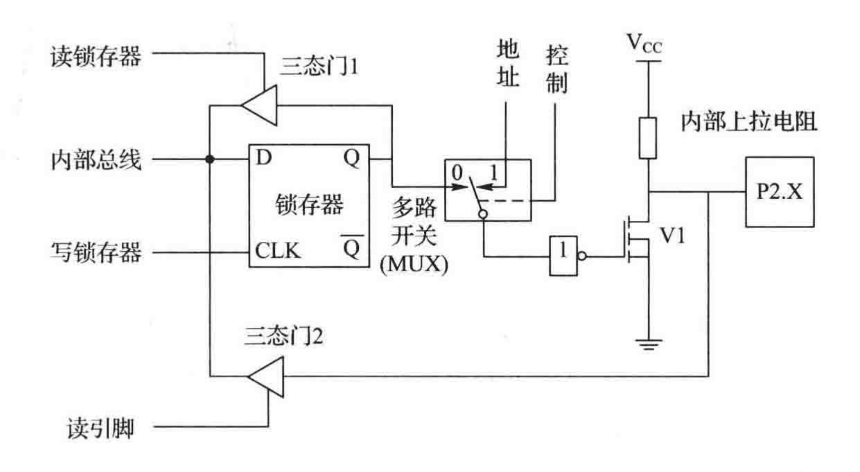


图2-11　P2口的内部电路结构

2. P2口的功能

1）通用I/O接口

当P2口作为通用I/O口使用时，多路开关(MUX)打向锁存器的输出端Q，构成一个准双向口。其功能与P1口相同，有输入、输出工作方式。

2）地址总线

P2口的另一种功能是作为系统扩展的地址总线口。当计算机从片外ROM中取指

令，或者执行访问片外 RAM、片外 ROM 的指令时，多路开关打在右边，P2 口上出现程序计数器(PC)的高 8 位地址或数据指针(DPTR)的高 8 位地址(A7～A15，低 8 位地址由 P0 输出)。上述情况下，锁存器的内容不受影响。所以，取指或访问外部存储器结束后，由于模拟开关打向左边，使输出驱动器与锁存器 Q 端相连，引脚上将恢复原来的数据。

一般来说，如果系统扩展了片外 ROM，取指的操作将连续不断，P2 口不断送出高 8 位地址，这时 P2 口就不应再作为通用 I/O 口使用。如果系统扩展了片外 RAM，需要由 P2 口、P0 口送出 16 位地址，则 P2 口也不再作为通用 I/O 接口。

3. 负载能力

P2 口的负载能力和 P1 口的相同，输出时能驱动 4 个 LSTTL 输入。

2.4.4 P3 口

1. P3 口的结构

图 2-12 所示是 P3 口其中 1 位的结构原理图，P3 口由 8 个这样的电路组成。

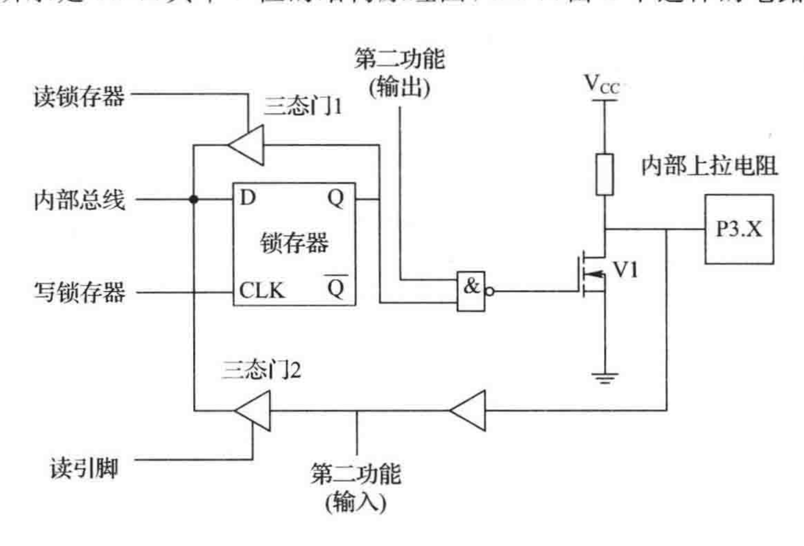


图 2-12　P3 口的内部结构

图中的锁存器起输出锁存作用。P3 口的 8 个锁存器组成特殊功能寄存器，场效应管 V1 与上拉电阻组成输出驱动器，以增大负载能力。三态门 2 是输入缓冲器，三态门 1 在端口操作时使用，与非门在端口作为第二功能时使用。

2. P3 口的功能

1) 通用 I/O 接口

MCS-51 单片机的 P3 口为多功能口。当第二功能输出端保持高电平时，与非门对锁存器 Q 端是畅通的，这时，P3 口实现第一功能，可作为通用 I/O 口使用，而且是一个准双向 I/O 口，其功能与 P1 口的相同。

2) 第二功能

P3 口除了作为准双向通用 I/O 接口使用外，每一根线还具有第二种功能，如表 2-5 所示。

表 2-5　P3 口各位线与第二功能

引脚	第二功能
P3.0	RxD(串行口输入)
P3.1	TxD(串行口输出)
P3.2	$\overline{INT0}$(外部中断 0 输入)
P3.3	$\overline{INT1}$(外部中断 1 输入)
P3.4	T0(定时器/计数器 0 的外部输入)
P3.5	T1(定时器/计数器 1 的外部输入)
P3.6	$\overline{WR}$(片外数据存储器“写选通控制”输出)
P3.7	$\overline{RD}$(片外数据存储器“读选通控制”输出)

当将第二功能作为专用信号输出时，锁存器 Q 端必须为高电平，否则场效应管 V1 导通，引脚被钳位在低电平，无法输入或输出第二功能信号。当锁存器置 1 时，与非门对第二输出功能是畅通的，但对输入而言，无论该位是作通用输入口还是作第二功能输入口，相应的输出锁存器和第二功能输出端都应置 1。实际上，由于 MCS-51 单片机所有 I/O 口的锁存器在上电复位时均被置 1，自然能满足上述要求，因此用户不必做任何操作，就可以直接使用 P3 口的第二功能。要在确信某一引脚第二功能所提供的信号不被使用(或不会产生)时，该引脚才可以作为 I/O 口使用。

在图 2-12 下方的输入通道中有两个缓冲器。第二功能的专用输入信号取自第一个缓冲器的输出端，而通用输入信号取自“读信号”缓冲器的输出端。

3. 负载能力

P3 口的负载能力和 P1 口的相同，输出时能驱动 4 个 LSTTL 负载。

2.5　单片机的辅助电路

辅助电路是单片机正常工作的必要条件。**单片机辅助电路主要有时钟电路和复位电路**。时钟电路给单片机提供时钟脉冲，保证单片机按照自身的时序自动工作起来；复位电路能对单片机进行初始化操作。对于 AT89 系列单片机，只要加入了正确的时钟电路和复位电路，就能构成单片机最小系统，即保证单片机系统正常工作的最简系统。

2.5.1　时钟电路

单片机是一个典型的时序电路器件，需要时钟电路提供时钟脉冲以保证其按照“节拍”正常工作。振荡器产生的信号送到 CPU，作为 CPU 的时钟信号，驱动 CPU 产生执行指令功能的机器周期。

MCS-51 单片机片内有一个由高增益反相放大器所构成的振荡电路，XTAL1 和 XTAL2 分别为振荡电路的输入和输出端，时钟可以由内部方式产生或由外部方式产生。

1. 内部方式

内部方式是通过外接石英晶体器件和内部振荡电路共同形成时钟电路。如图 2-13 所示，在 XTAL1 和 XTAL2 引脚上外接定时元件，内部振荡电路就产生自激振荡。定时元件通常采用石英晶体和电容组成的并联谐振回路。晶振频率可以在 1.2～24 MHz 之间选择，电容 C1 和 C2 的值为 10～30 pF，时钟频率基本上由晶振决定，电容的大小可起频率微调作用。

2. 外部方式

外部方式是把外部已有的时钟信号引入单片机内，即把外部振荡器的信号直接连到 XTAL1 端，XTAL2 端悬空不用，如图 2-14 所示。对外部时钟信号无特殊要求，只要保证脉冲宽度，一般采用频率低于 12 MHz 的方波信号。采用外部方式的好处是，可以通过外部时钟频率控制来改变单片机的机器周期，以降低电磁干扰(EMI)。一般应用很少使用外部方式。

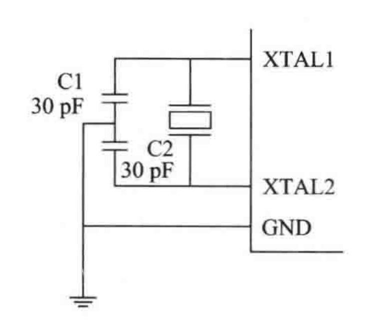


图 2-13　使用内部振荡器的晶振连接

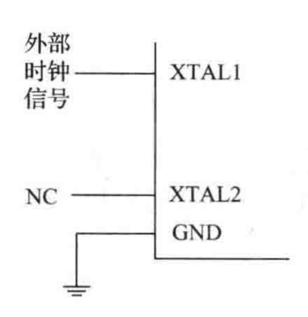


图 2-14　使用外部时钟的连接

2.5.2　复位电路和复位状态

MCS-51 系列单片机与其他微处理器一样，在启动时都需要复位，使 CPU 及系统各部件处于确定的初始状态，并从初始状态开始工作。在单片机系统设计并制作完成后，就要上电工作，在上电初期，由于单片机内部电压不稳定，程序执行会混乱，因而要等到电压稳定后才让单片机进行工作。同时，在单片机工作期间，由于外界干扰或其他原因使系统工作不正常，就需要进行上电复位和手动复位操作。如果 RST 引脚上有一个高电平并**维持 2 个机器周期(24 个振荡周期)或更多，则 CPU 可响应并将系统复位。**

实现可靠的复位需要正确的复位电路，而复位后单片机将处于怎样的初始状态，这是单片机应用系统开发者必须掌握的。

1. 复位电路

单片机的复位可以通过多种方式实现，对应着不同的复位电路。复位的方法有三种，即上电复位、手动开关复位和 WDT(看门狗)复位。

1) 上电复位电路

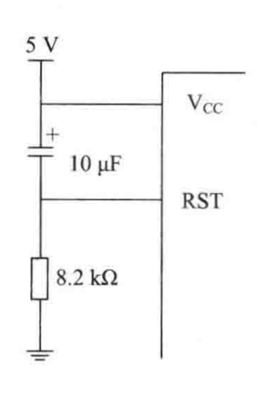


图 2-15　上电复位电路

上电复位电路如图 2-15 所示。只要在复位输入引脚 RST 上接一个电容至 V_{CC} 端，下接一个电阻到地即可。对于 CMOS 型单片机，由于在 RST 端内部有一个下拉电阻，故可将外部电

阻去掉，而将外接电容减至 1 μF。

上电复位的过程是在加电时，复位电路通过电容加给 RST 端一个短暂的高电平信号，此高电平信号随着 V_{CC} 对电容的充电过程而逐渐回落，即 RST 端的高电平持续时间取决于电容的充电时间。为了保证系统能够可靠地复位，RST 端的高电平信号必须维持足够长的时间。

上电时，V_{CC} 的上升时间约为 10 ms，而振荡器的起振时间取决于振荡频率。如晶振频率为 10 MHz，起振时间为 1 ms；晶振频率为 1 MHz，起振时间则为 10 ms。

如果系统在上电时得不到有效的复位，则在程序计数器(PC)中将得不到一个合适的初值，从而导致 CPU 可能会从一个未被定义的位置开始执行程序。

2）手动复位电路

手动复位电路是上电复位和手动复位相结合的，主要用于单片机系统故障(死机)时的重新启动。可以人为地在复位输入端 RST 上加入高电平，一般采用的办法是在 RST 端和正电源 V_{CC} 之间接一个按钮。当按下按钮时，V_{CC} 的 +5 V 电平就会直接加到 RST 端，虽然按下按钮的时间很短，但是也会使按钮保持接通数十毫秒的时间，所以手动复位能满足复位的时间要求。手动复位电路如图 2 - 16 所示。

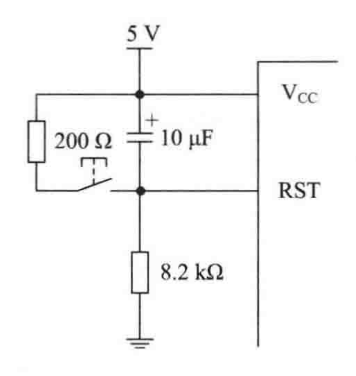


图 2 - 16 手动复位电路

3）WDT 复位电路

WDT 复位电路(看门狗复位电路)是利用 MAX705 等 WDT 专用芯片来实现复位的电路，如图 2 - 17 所示。WDT 芯片内有一个不受外部控制的计数器，上电后即自动计数，一旦计数溢出就发出对单片机的复位信号。为了不使计数器溢出，必须在计数器溢出前通过 WDI 口输入清 0 信号，使计数器复位清 0。因而在实际运用中，只要在软件上适当安排，即可使得单片机处于正常工作状态。针对图 2 - 17 所示电路的正常程序序列中应能循环执行到“CPL　P3.7”取反指令(详见第 3 章)，从而使 WDT 内部的计数器能及时清 0，保证了单片机系统的正常运行。当有外部干扰因素出现，使得单片机程序工作不正常时，便无法确保指令“CPL　P3.7”的执行，WDT 芯片内部的计数器就会产生溢出，$\overline{\text{WDO}}$变成低电平，进而使得$\overline{\text{MR}}$变低，单片机系统自动复位，从 0000H 重新开始工作。

实际应用中，我们通常把手动复位电路和 WDT 复位电路结合起来使用。

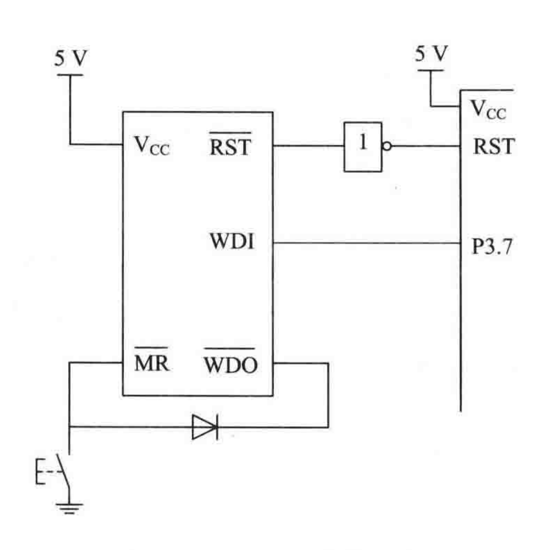


图 2-17　WDT 复位电路

2. 复位状态

系统复位后，许多特殊功能寄存器都将恢复到初始状态。各特殊功能寄存器的状态如表 2-6 所示，其中×为随机数。

手动复位或 WDT 复位后，片内 RAM 和片外 RAM 的内容保持不变，但在上电复位后为随机数。

表 2-6　系统复位后特殊功能寄存器的状态

寄存器	复位后的内容	寄存器	复位后的内容
PC	0000H	TH0	00H
ACC	00H	TL0	00H
B	00H	TH1	00H
PSW	00H	TL1	00H
SP	07H	IP	×××00000B
DPTR	0000H	IE	0××00000B
P0～P3	FFH	SCON	00H
TMOD	00H	SBUF	不定
TCON	00H	PCON	0×××0000B

2.5.3　单片机最小系统

最小应用系统是指能维持单片机运行的最简单配置的系统。由于 89C51 单片机有片内 ROM，所以其最小应用系统即为配有时钟电路、复位电路和电源的单个单片机，如图 2-18所示。由于资源的限制，最小应用系统只能用作一些小型的控制单元。

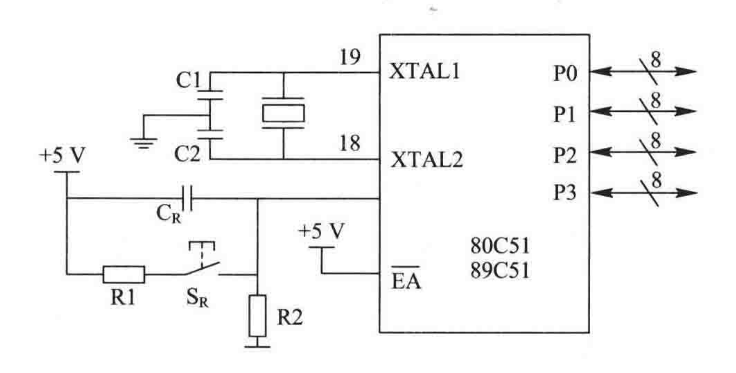


图 2-18　80C51/89C51 最小应用系统

2.6　单片机的工作时序和工作方式

时序是单片机指令执行中各信号之间的相互关系。单片机本身如同一个复杂的同步时序逻辑电路，时钟电路用于产生单片机工作所需要的时钟信号。为了保证同步工作方式的实现，单片机内部电路应在唯一的时钟信号控制下严格地按时序进行工作。

2.6.1　时序的基本概念

1. 时钟周期

时钟周期也称振荡周期，是指为单片机提供时钟信号的振荡源的周期或外部输入时钟的周期。考虑到绝大部分单片机应用系统是采用石英晶体作为振荡源，一般来讲，时钟周期也就是 $1/f_{osc}$，f_{osc}是石英晶体的振荡频率(简称晶振频率)。

2. 机器周期

完成一条指令的一个基本操作步骤所需的时间称为机器周期。一个机器周期由 6 个状态组成，即 S1～S6，每个状态又被分成两个节拍 P1 和 P2，如图 2-19 所示。所以**一个机器周期有 12 个振荡周期**，可以依次表示为 S1P1，S1P2，…，S6P1，S6P2。如果石英晶体振荡频率 $f_{osc}=12$ MHz，则机器周期为$(1/f_{osc})\times 12=1\ \mu s$。单片机的某些单周期指令的执行时间就是一个机器周期。

3. 指令周期

单片机 CPU 执行一条指令所需的时间称为指令周期。MCS-51 系列单片机执行不同指令所需时间也不尽相同，有单机器周期、双机器周期、四机器周期三种指令周期。如："MOV　A，# data"(把一个数写入累加器中)就是一个单机器周期指令。附录 B 中给出了每条指令的机器周期数。

2.6.2　单片机的工作时序

MCS-51 单片机指令按照执行时间分为三类：单机器周期指令(简称单周期指令)、双机器周期指令(简称双周期指令)和四机器周期指令(简称四周期指令)。而按照指令占用存储空间长度分，又有单字节指令、双字节指令和三字节指令(详见第 3 章)。所以有以下几种情况：

- 单字节单周期指令；

- 单字节双周期指令；
- 双字节单周期指令；
- 双字节双周期指令；
- 三字节双周期指令；
- 单字节四周期指令。

图 2-19 给出了 89C51 单片机的取指和执行指令的时序关系。这些内部时钟信号不能从外部观察到，我们用 XTAL1 振荡信号作参考。从图中可以看到，低 8 位地址的锁存信号 ALE 在每个机器周期中两次有效：一次在 S1P2 与 S2P1 期间，另一次在 S4P2 与 S5P1 期间。这说明在一个机器周期内有两次取指操作。

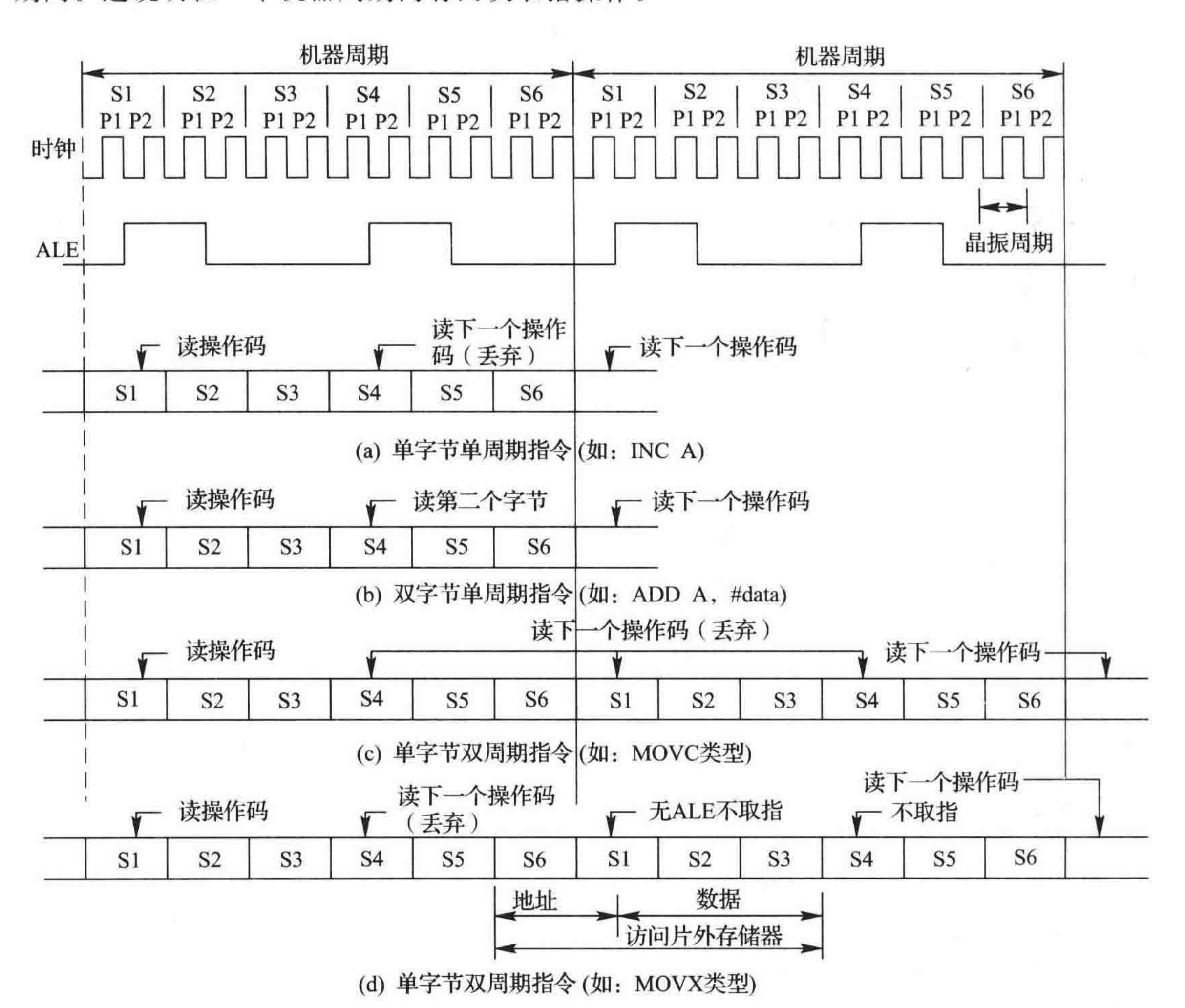


图 2-19　单片机取指和执行指令的时序关系

对于单周期指令，当操作码被送入指令寄存器时，便从 S1P2 开始执行指令。如果是双字节单周期指令，则在同一机器周期的 S4 期间读入第二个字节；如果是单字节单周期指令，则在 S4 期间仍进行读操作，但所读的这个字节操作码被忽略，程序计数器也不加 1，在 S6P2 结束时完成指令操作，如图 2-19(a)所示。对于双字节单周期指令，通常是在一个机器周期内从程序存储器中读入两个字节，如图 2-19(b)所示。唯有 MOVX 指令例外，MOVX 是访问外部数据存储器的单字节双周期指令。在执行 MOVX 指令期间，外部

数据存储器被访问且被选通时跳过两次取指操作，如图 2－19(d)所示。图 2－19(c)给出了一般情况下单字节双周期指令的时序。MCS－51 指令大部分在一个机器周期或两个机器周期内完成。乘(MUL)和除(DIV)指令是仅有的需要两个以上机器周期的指令，占用 4 个机器周期。

2.6.3　单片机的工作方式

1. 正常工作方式

当单片机完成复位后，进入正常工作方式，这时单片机由 V_{CC}供电。正常工作方式是单片机自动完成任务的工作方式。正常工作过程是单片机执行程序的过程，即一条条执行指令的过程。

程序通常是顺序执行的，所以程序中的指令也是一条一条地顺序存放的。单片机在执行程序时要能把这些指令一条一条地从 ROM 中取出并加以执行，必须通过一个部件追踪指令所在的地址，这一部件就是程序计数器(PC，包含在 CPU 控制器中)。在开始执行程序时，给 PC 赋以程序中第一条指令所在的地址，然后取得每一条要执行的命令，每次读操作码或操作数时，PC 中的内容就会自动加 1，然后执行指令。本条指令执行完毕后，PC 指向下一条指令的起始地址，保证指令顺序执行。

2. 掉电工作方式

MCS－51 系列单片机的 SFR 中有一个电源控制寄存器(PCON)，地址为 87H。PCON 寄存器的控制格式如下：

	D7	D6	D5	D4	D3	D2	D1	D0
(87H)								
PCON	SMOD	—	—	—	GF1	GF0	PD	IDL

PCON 各位可以进行读/写操作。PCON 不能进行位操作，只能按字节操作。

SMOD：波特率加倍位(用途见第 6 章)。

GF1、GF0：通用标志位。

PD：掉电方式控制位。当 PD 位为 1 时，启用掉电方式。

IDL：待机方式控制位。当 IDL 位为 1 时，启用待机模式。

1) 进入掉电工作方式

当单片机检测到电源故障时，立即通过外部中断引脚中断运行程序。CPU 转而执行中断服务程序，首先进行信息保护，然后执行一条置 PD 位为 1 的指令，系统进入掉电工作方式。在这种工作方式下，内部振荡器停止工作。由于没有振荡时钟，因此所有的功能部件都停止工作，但内部 RAM 区和特殊功能寄存器的内容被保留，而端口的输出状态值都保存在对应的 SFR 中，ALE 和$\overline{PSEN}$都为低电平。

2) 退出掉电工作方式

退出掉电方式的唯一方法是硬件复位。复位后将所有的特殊功能寄存器的内容初始化，但不改变内部 RAM 区的数据。

在掉电工作方式下，V_{CC}可以降到 2 V，但在进入掉电方式之前，V_{CC}不能降低。而在准备退出掉电方式之前，V_{CC}必须恢复到正常的工作电压值，并维持一段时间(约 10 ms)，

使振荡器重新启动并稳定后，方可退出掉电方式。

3. 低功耗工作方式

1）进入低功耗工作方式

低功耗工作方式也称为待机工作方式。当CPU执行完置IDL位为1的指令后，系统进入待机工作方式。这时，内部时钟通向CPU的电路被阻断，而只供给中断、串行口和定时器。CPU的内部状态维持不变，即包括堆栈指针(SP)、程序计数器(PC)、程序状态字(PSW)、累加器(ACC)等所有的内容保持不变，端口状态也保持不变。ALE和$\overline{\text{PSEN}}$保持逻辑高电平。

由于CPU耗电量占单片机芯片耗电量的80%～90%，因此CPU停止工作会大大降低功耗。

2）退出低功耗工作方式

进入待机工作方式后，有两种方法可以使系统退出待机工作方式。

一种方法是任何的中断请求都可以由硬件将IDL清0而中止待机工作方式。引入一个外部中断，CPU响应中断的同时，IDL被硬件自动清0，CPU进入中断服务程序，执行到RETI指令时，结束中断，返回主程序，进入正常工作方式。PCON寄存器中的GF0和GF1标志可用来指示中断是在正常方式下还是在待机方式下发生。

另一种方法是硬件复位，需要在RST引脚加入正脉冲。由于在待机工作方式下振荡器仍然工作，因此硬件复位仅需2个机器周期以上便可完成。而RST端的复位信号直接将IDL清0，从而退出待机状态，CPU则从进入待机方式的下一条指令开始重新执行程序，进入正常工作方式。

思考与练习

1. MCS-51单片机内部包含哪些主要功能部件？

2. MCS-51单片机内部数据存储器可以分为哪几个不同区域？它们之间有什么区别？

3. 程序状态字寄存器(PSW)的作用是什么？常用状态有哪些位？作用是什么？

4. 位地址6DH与字节地址6DH如何区分？位地址6DH对应的字节地址单元是什么？

5. 如何选择当前工作寄存器组？

6. 什么是堆栈？堆栈的特点是什么？堆栈指针(SP)的作用是什么？复位后，SP是多少？堆栈实际上是从什么地方开始的？

7. 简述$\overline{\text{EA}}$、ALE、$\overline{\text{PSEN}}$、$\overline{\text{RD}}$、$\overline{\text{WR}}$引脚的用途。

8. MCS-51单片机I/O端口有什么特点？作为输出口时如何使用？作为输入口时又如何使用？

9. MCS-51单片机控制总线信号有哪些？作用是什么？

10. MCS-51单片机复位有几种方法？复位后各寄存器处于什么状态？

11. WDT复位电路的工作原理是什么？

12. 什么是振荡周期、机器周期、指令周期和时钟周期？它们之间有何关系？当晶振频率为6 MHz时，一个机器周期是多少？

13. MCS-51有几种低功耗方式？如何实现？

第 3 章　单片机指令系统

本章重点讲述 MCS－51 单片机指令的格式、寻址方式、指令系统，并针对常用指令给出相应的应用程序。

3.1　指令系统概述

指令是计算机(单片机)按照人们的意图来执行某种操作的命令。例如，若要使计算机完成两个数的加法操作，就要使计算机执行加法指令。一台计算机能够执行的全部指令的集合称为该计算机的指令系统。指令系统的功能强弱在很大程度上决定了计算机性能的高低。

指令一般有功能、时间和空间三种属性。功能属性是指每条指令都对应一个特定的操作功能；时间属性是指一条指令执行所用的时间，一般用机器周期来表示；空间属性是指一条指令在 ROM 中存储所占用的字节数。

MCS－51 单片机指令系统共有 111 条基本指令，具有功能强、指令短、执行快等特点。按功能属性划分，MCS－51 指令系统可分为以下五类：

(1) 数据传送类指令(29 条)；

(2) 算术运算类指令(24 条)；

(3) 逻辑运算类指令(24 条)；

(4) 控制转移类指令(17 条)；

(5) 位操作类指令(17 条)。

按空间属性划分，MCS－51 指令系统可分为以下三类：

(1) 单字节指令(49 条)；

(2) 双字节指令(45 条)；

(3) 三字节指令(17 条)。

按时间属性划分，MCS－51 指令系统可分为以下三类：

(1) 单机器周期指令(64 条)；

(2) 双机器周期指令(45 条)；

(3) 四机器周期指令(只有乘、除法 2 条指令)。

由此可见，MCS－51 单片机指令系统在存储空间和执行时间方面具有较高的优势。

3.1.1　指令的表达形式

通常，指令有两种表达形式：机器语言形式和汇编语言形式。机器语言指令用二进制代码表示每条指令，是计算机能直接识别和执行的指令。但是用机器语言编写程序不方便记忆，不容易阅读、理解和查错。为方便程序的编写，人们采用一种助记符号来反映指令

的功能和主要特征，这种用助记符表示的机器指令被称为汇编语言指令。汇编语言指令直观、易记忆、好理解、易阅读，执行速度快，实时性强，且与计算机的机器语言指令一一对应，因此单片机系统开发时可以采用汇编语言指令。即使使用C语言，也需要了解汇编语言指令。

MCS-51单片机汇编语言指令的一般格式如下：

```
[标号：] 操作码 [操作数] [；注释]
```

即一条汇编语言指令可以由标号、操作码、操作数和注释四部分组成，其中操作码是指令中必不可少的内容，其余三部分可以根据实际情况取舍，因此在格式中使用了可选择符号“[ ]”。下面对组成指令的四个部分分别进行介绍。

① 标号位于语句的开始，由字母和数字组成，它代表该指令的地址，也称为指令的符号地址。标号必须以字母开始，其余部分可以是字母、数字和符号，但不能为MCS-51单片机指令集中的指令助记符。标号与操作码之间必须用冒号“：”分隔。

② 操作码是指令的助记符(表明指令功能的英语单词或其缩写)，表示指令所执行的操作功能，描述指令的操作性质，是一条指令中唯一不能空缺的部分。

③ 操作数为指令的操作对象，既可以是参加操作的数据，也可以是操作数所在的地址。MCS-51单片机的指令可以没有操作数，也可以有1～3个操作数。多个操作数之间必须用逗号“,”分隔，操作数与操作码之间要用空格分隔。

操作数中的数据可以是二进制、十进制或十六进制数据，也可以是ASCII码，具体格式如下：

二进制数，以字母B结尾，如10101010B。

十进制数，以字母D结尾或将D省略，如78D或78。

十六进制数，以字母H结尾，如78H、0A2H。注意：十六进制数以A～F开头时应在其前面加上数字“0”。

ASCII码，字符外加单引号''标志，如'A'。

④ 注释位于语句的最后，是说明语句功能和性质的文字。注释的主要作用是对程序段或者某条指令在整个程序中的作用进行解释和说明，以帮助阅读、理解和使用源程序。操作码与注释之间可以用分号“；”分隔，在Keil μVision集成环境中，也可以用“//”分割。

注意：**指令中的标点符号均为英文符号。**

由上述分析可知，在MCS-51单片机的指令系统中，**指令主要由操作码和操作数组成，而操作码和操作数都有对应的二进制代码。**通常，操作码用1个字节表示，操作数用1个或2个字节表示。因此MCS-51单片机的指令按二进制代码所占的字节数可以分为单字节指令、双字节指令和三字节指令三种格式。

1. 单字节指令

单字节指令中的8位二进制代码既包含操作码的信息，也包含操作数的信息，这种指令包含以下三种情况：

① 无操作数指令，如指令“NOP”的编码为0000 0000B，其十六进制表示形式为00H，该编码仅为操作码，说明指令实现空操作功能。

② 指令码中隐含着对某一寄存器的操作，如指令“INC　A”的编码为0000 0100B，其

十六进制表示形式为04H，该编码仅为操作码，而操作数(累加器A)隐含在操作码中，该指令的功能是将累加器A的内容加1。

③ 指令码含有操作码和寄存器编码，如指令“MOV A，Rn”的编码为1110 1rrrB，该编码的高5位为操作码，低3位rrr为存放操作数的寄存器编码，与n的值相对应，该指令的功能是将当前工作寄存器Rn(n=0～7)中的数据传送到累加器A中。

2. 双字节指令

双字节指令用一个字节表示操作码，另一个字节表示操作数或操作数所在的地址，如指令“MOV A，#40H”的代码(十六进制形式)为74H、40H，其中74H为操作码，40H为操作数，该指令的功能是将立即数“40H”传送到累加器A中。

3. 三字节指令

三字节指令用一个字节表示操作码，两个字节表示操作数或操作数所在的地址，如指令“MOV 50H，#40H”的代码(十六进制形式)为74H、50H、40H，其中74H为操作码，50H为数据存放的地址，40H为操作数，该指令的功能是将立即数“40H”传送到片内RAM的50H单元中。

3.1.2 指令中的常用符号

在描述指令系统和寻址方式时，常用一些符号来代表具体的操作数，下面简单介绍这些常用符号的意义。

Rn：当前选中的寄存器组中的8个工作寄存器R0～R7(n=0～7)。

Ri：当前选中的寄存器组中的2个工作寄存器R0和R1(i=0，1)。

@：间接寻址寄存器或基址寄存器的前缀，如@Ri，@DPTR。

#data：8位立即数，即包含在指令中的8位常数，00H～FFH。

#data16：16位立即数，即包含在指令中的16位常数，0000H～FFFFH。

direct：8位直接地址，既可以是片内RAM的低128字节的单元地址，也可以是SFR的单元地址或名称(包括I/O口P0、P1、P2和P3)。

addr11：表示11位目的地址，主要用于ACALL和AJMP指令中。

addr16：表示16位目的地址，主要用于LCALL和LJMP指令中。

rel：补码形式的8位地址偏移量，取值范围为－128～＋127，主要用于相对转移指令，以形成转移的目的地址。

bit：表示片内RAM或SFR内的直接寻址位。

A或ACC：表示累加器。

B：表示B寄存器。

C：表示位累加器。

DPTR：表示数据指针寄存器，可用作16位地址寄存器。

$：表示当前指令的地址。

X：表示直接地址或寄存器。

(X)：表示X地址单元或寄存器中的内容。

((X))：表示以X地址单元或寄存器的内容为地址所指定单元的内容。

/：在位操作指令中，表示对该位求反后再参与操作。

←：表示指令的操作结果是将箭头右边的内容传送到箭头的左边。

→：表示指令的操作结果是将箭头左边的内容传送到箭头的右边。

3.2 单片机的寻址方式

计算机在执行指令时，要先寻找参加运算的操作数，然后对其进行操作并将操作结果存入相应存储单元或寄存器。我们把寻找源操作数所在单元的地址称为寻址，而寻找源操作数所在地址的方法就是寻址方式。如：指令“MOV A，R0”中，第一个操作数“A”称为目的操作数，第二个操作数“R0”称为源操作数，寻址方式就是寻找“R0”所在地址的方法。

寻址方式是计算机性能的具体体现，也是编写汇编语言程序的基础，开发者必须非常熟悉并能灵活运用它。寻址方式的种类越多，计算机的功能越强，灵活性越大，处理数据的效率也就越高。

在 MCS-51 单片机中，操作数的存放范围是很宽的，可以放在片外 ROM/RAM 中，也可以放在片内 ROM/RAM 以及 SFR 中。为了适应在操作数范围内的寻址，MCS-51 的指令系统共使用了七种寻址方式，它们是立即寻址、直接寻址、寄存器寻址、寄存器间接寻址、变址寻址、相对寻址和位寻址，每种寻址方式所对应的寄存器和存储空间如表 3-1所示。

表 3-1 七种寻址方式所对应的寄存器和存储空间

寻址方式	使用的变量	寻址空间
立即寻址	直接给出数值，无变量	ROM
直接寻址	直接给出地址，无变量	片内 RAM 低 128 字节，SFR
寄存器寻址	R0～R7、A、B、DPTR、位累加器 C	工作寄存器 R0～R7，部分 SFR
寄存器间接寻址	@R0、@R1、SP 或@DPTR	片内 RAM 或片外 RAM
变址寻址	@A+DPTR、@A+PC	ROM
相对寻址	PC+偏移量	ROM
位寻址	直接给出位地址或位符号	片内 RAM 的位寻址区，SFR 的可寻址位

3.2.1 立即寻址

在指令中直接给出操作数的寻址方式称为立即寻址。在指令编码中，操作数紧跟在操作码后面，与操作码一起存放在指令代码段(ROM)中，这样的操作数称为立即数。采用汇编语言编写指令时，**把“#”号放在立即数前面，以表示该寻址方式为立即寻址。**

【例 3-1】 有一指令为“MOV A，#68H”，该指令执行后，(A)=68H，执行过程如图 3-1 所示。

指令“MOV A，#68H”在 ROM 中的编码为 01110100B、01101000B，表示成十六进制形式为 74H、68H。指令执行时，CPU 在对操作码进行分析后得知该指令的功能是将操作码后面的 8 位立即数“68H”复制到累加器 A 中，所以指令执行后累加器 A 的内容变为“68H”，而 A 中原来的内容被覆盖。

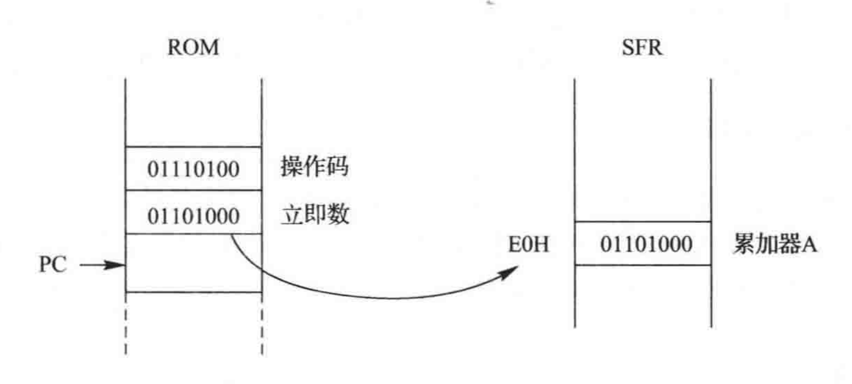


图 3-1　指令“MOV　A，#68H”的执行示意图

在 MCS-51 单片机指令系统中，仅有一条包含 16 位立即数的指令，形式为“MOV　DPTR，#data16”，其中“#data16”表示 16 位立即数。如：指令“MOV　DPTR，#1234H”，其功能是把 16 位立即数“1234H”传送到寄存器 DPTR 中，其中高 8 位“12H”送到 DPH，低 8 位“34H”送到 DPL。

因为立即数直接存放在 ROM 中，所以立即寻址所对应的寻址空间为 ROM 空间。

3.2.2　直接寻址

指令中直接给出操作数所在存储单元地址的寻址方式称为直接寻址。在指令编码中，操作数所在存储单元的地址紧跟在操作码之后，与操作码一起存放在指令代码段中，而操作数本身则存放在该地址所指示的存储单元中。

【例 3-2】　如果片内 RAM 中(30H)=56H，则指令“MOV　A，30H”执行后，(A)=56H，执行过程如图 3-2 所示。

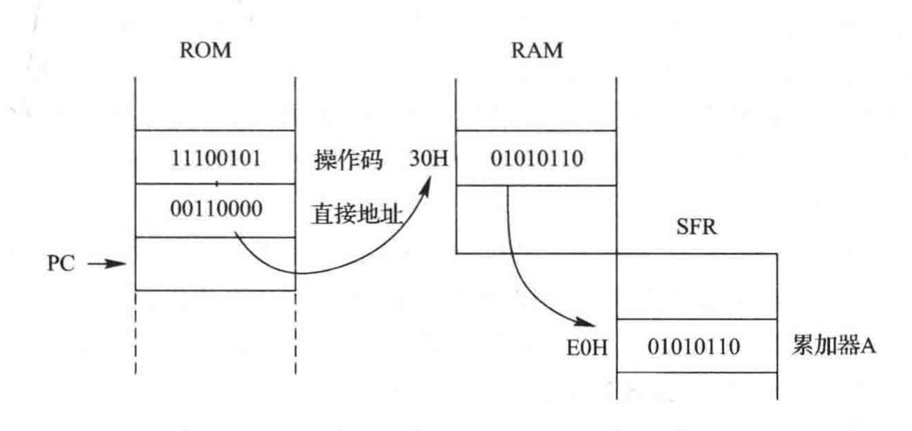


图 3-2　指令“MOV　A，30H”的执行示意图

指令“MOV　A，30H”在 ROM 中的编码(十六进制形式)为 E5H、30H。指令执行时，CPU 在对操作码进行分析后得知该指令的功能是将 30H 单元的内容复制到累加器 A 中，A 中原来的内容被覆盖。因为 30H 单元的内容为“56H”，所以指令执行后累加器 A 的内容变为“56H”。

直接寻址方式只能用来访问片内 RAM 的低 128 字节和 SFR 区域。对于片内 RAM 的低 128 字节，在指令中以直接地址形式表示操作数所在单元的地址，如“MOV　A，direct”，其中“direct”代表直接地址。对于 SFR，在指令中既可以用单元地址也可以用寄存

器符号来表示操作数所在单元的地址。如指令"MOV　A，80H"可以写成"MOV　A，P0"，这里的"P0"与地址"80H"是等同的，但"P0"这种写法更容易理解和阅读。

3.2.3 寄存器寻址

操作数存放在寄存器中，指令中直接给出该寄存器名称的寻址方式称为寄存器寻址。存放操作数的寄存器在指令代码中不占据单独的一个字节，而是包含在操作码字节中。寄存器寻址一般用于访问选定的工作寄存器组的 8 个工作寄存器 R0～R7，并以符号名称来表示寄存器。

【例 3-3】 如果(R1)=80H，则指令"MOV　A，R1"执行后，(A)=80H，执行过程如图 3-3 所示。

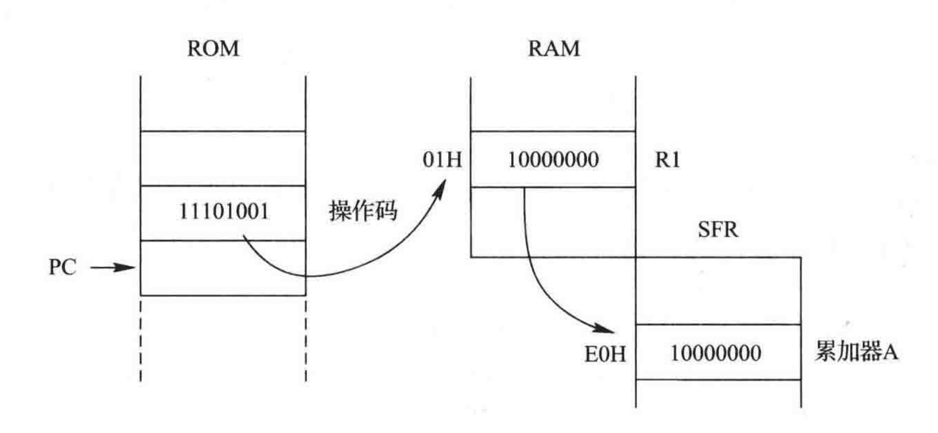


图 3-3　指令"MOV　A，R1"的执行示意图

指令"MOV　A，R1"在 ROM 中的编码为 11101 001 B，其中低 3 位 001 表示 R1 寄存器。指令执行时，CPU 在对操作码进行分析后得知该指令的功能是将 R1 寄存器的内容复制到累加器 A 中，A 中原来的内容被覆盖。由表 2-2 可知，Rn(n=0～7)的物理地址由 PSW 中的 RS1 和 RS0 状态决定。因此，若假设此时 RS1 RS0=00B，则可知 R1 属于第 0 组工作寄存器，其地址为 01H。因为 01H 单元的内容为"80H"，所以指令执行后累加器 A 的内容变为"80H"。

除了工作寄存器 R0～R7 符合寄存器寻址条件以外，部分特殊功能寄存器如累加器 A、寄存器 B(以 AB 寄存器对形式出现)以及数据指针 DPTR 也可以采用寄存器方式寻址。

需要注意的是，指令"MOV　Rn，A"属于寄存器寻址，而指令"MOV　Rn，ACC"属于直接寻址，因为符号 ACC 代表累加器的单元地址"E0H"，但二者实现的功能是一样的。

此外，对于第 0 组工作寄存器，R0 的物理地址为 00H，因此指令"MOV　A，00H"和"MOV　A，R0"实现的功能是一样的，都是将 00H 单元的内容复制到累加器 A 中。但是两者也是有区别的："MOV　A，00H"属于直接寻址，机器码为 E5H、00H，即这条指令要占用两个字节；"MOV　A，R0"属于寄存器寻址，机器码为 E8H，即这条指令只占用 1 个字节。由此可见，同一个功能可以采用不同的指令来实现，而且对于同一个存储单元，若采用不同形式表示，所对应的寻址方式是不同的。

3.2.4 寄存器间接寻址

指令中给出的寄存器中存放的不是操作数，而是操作数所在单元的地址，类似C语言的指针，即操作数是通过指令中给出的寄存器间接得到的，这种寻址方式被称为寄存器间接寻址。为了与寄存器寻址区别，应在寄存器的名称前面加前缀标志“@”。

【例3-4】 如果(R0)=40H，(40H)=30H，则指令“MOV A，@R0”执行后，(A)=30H，执行过程如图3-4所示。

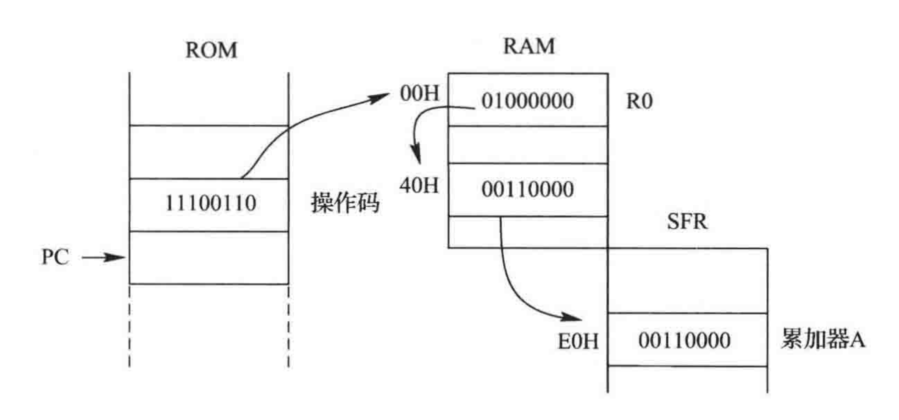


图3-4 指令“MOV A，@R0”的执行示意图

指令“MOV A，@R0”在ROM中的编码为11100110B，其中最低位0表示间接寻址寄存器为R0。指令执行时，CPU在对操作码进行分析后得知该指令的功能是以R0寄存器的内容作为操作数地址，然后将该地址单元的内容复制到累加器A中，A中原来的内容被覆盖。若假设此时RS1 RS0=00B，则可知R0属于第0组工作寄存器，其地址为00H。因为00H单元的内容为“40H”，则指令执行后，40H单元的数据“30H”被传送到累加器A中，所以A的内容变为“30H”。

采用寄存器间接寻址不仅可以访问片内RAM，还可以访问片外RAM，间接寻址寄存器可以采用寄存器R0或R1、数据指针DPTR和堆栈指针SP。应用时要注意以下几个问题：

(1) 访问片内RAM的低128字节，可以采用寄存器R0或R1作间接寻址寄存器。其通用形式为“MOV指令 @Ri(i=0，1)”。例如，指令“MOV A，@R1”，其功能是把R1指定的片内RAM单元的内容送入累加器A。

注意：增强型片内RAM的高128字节只能采用寄存器间接寻址方式。

(2) 访问片外RAM的64 KB，通常采用数据指针(DPTR)作间接寻址寄存器。如指令“MOVX A，@DPTR”，其功能是把DPTR指定的片外RAM单元的内容送入累加器A。

(3) 堆栈操作指令PUSH和POP使用堆栈指针SP作间接寻址寄存器来对堆栈区进行间接寻址。

3.2.5 变址寻址

变址寻址是以DPTR或PC作为基址寄存器，以累加器A作为变址寄存器，并以两者内容相加形成操作数所在单元的地址。

在 MCS－51 单片机中，用变址寻址方式只能访问 ROM，寻址范围为 64KB。变址寻址用于对 ROM 中的数据进行寻址，特别适用于查表。例如：

```
MOVC    A, @A+DPTR
```

此指令的功能是将 DPTR 的内容与 A 的内容相加，形成新的地址，再将此地址对应单元中的 8 位二进制数送入 A 中。

【例 3－5】 如果(DPTR)＝1234H，(A)＝60H，ROM 中(1294H)＝78H，则指令"MOVC　A，@A＋DPTR"执行后，(A)＝78H，执行过程如图 3－5 所示。

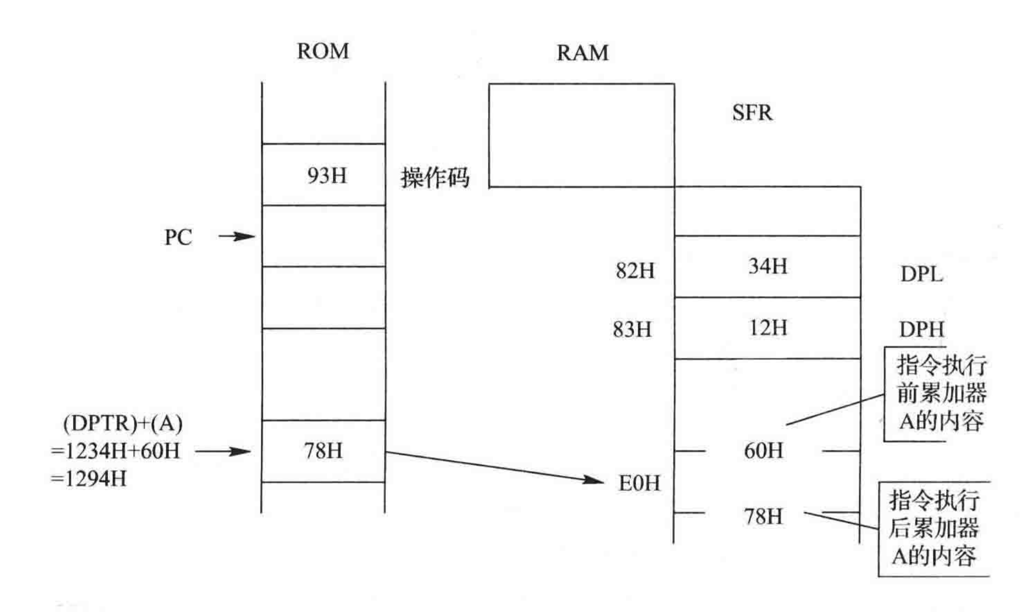


图 3－5　指令"MOVC　A，@A＋DPTR"的执行示意图

指令"MOVC　A，@A＋DPTR"在 ROM 中的编码(十六进制形式)为 93H。指令执行时，CPU 在对操作码进行分析后得知该指令的功能是将累加器 A 的内容与数据指针寄存器 DPTR 中的内容相加，并将相加的结果作为地址，再将该地址单元的内容送入累加器 A 中，A 中原来的内容被覆盖。因为累加器 A 的内容为"60H"，数据指针寄存器 DPTR 的内容为"1234H"，则二者相加得到地址"1294H"，然后将该地址单元的内容"78H"复制到累加器 A 中，所以指令执行后累加器 A 的内容变为"78H"。

3.2.6　相对寻址

相对寻址是以程序计数器 PC 的当前值加上指令中给出的偏移量 rel 形成目标地址的寻址方式。采用该寻址方式进行操作时修改的是 PC 值，因此这种寻址方式用于实现程序的分支跳转。

在使用相对寻址时需要注意以下两点：

(1) PC 的当前值是读出该相对转移指令(2 字节或 3 字节)后，PC 指向的下一条指令的地址，即

PC 的当前值＝相对转移指令所在存储单元的地址＋指令字节数

例如："JZ　rel "是一条累加器 A 为 0 就转移的双字节指令。若该指令的存储地址为

2010H，则执行该指令时的PC当前值为2012H，即PC当前值是对相对转移指令取指令结束时的值。

(2) 偏移量rel是一个有符号的8位二进制数，以补码形式置于操作码之后存放，取值范围是－128～＋127。负数表示向地址减小的方向转移，正数表示向地址增加的方向转移，相对的基准是PC的当前值。因此，转移的目的地址为

目的地址＝PC当前值＋rel＝指令存储地址＋指令字节数＋rel

【例3-6】 如果rel为75H，PSW.7为1，指令"JC rel"存放在0200H开始的单元，则执行"JC rel"指令后，程序将跳转到0277H单元取指令并执行，执行过程如图3-6所示。

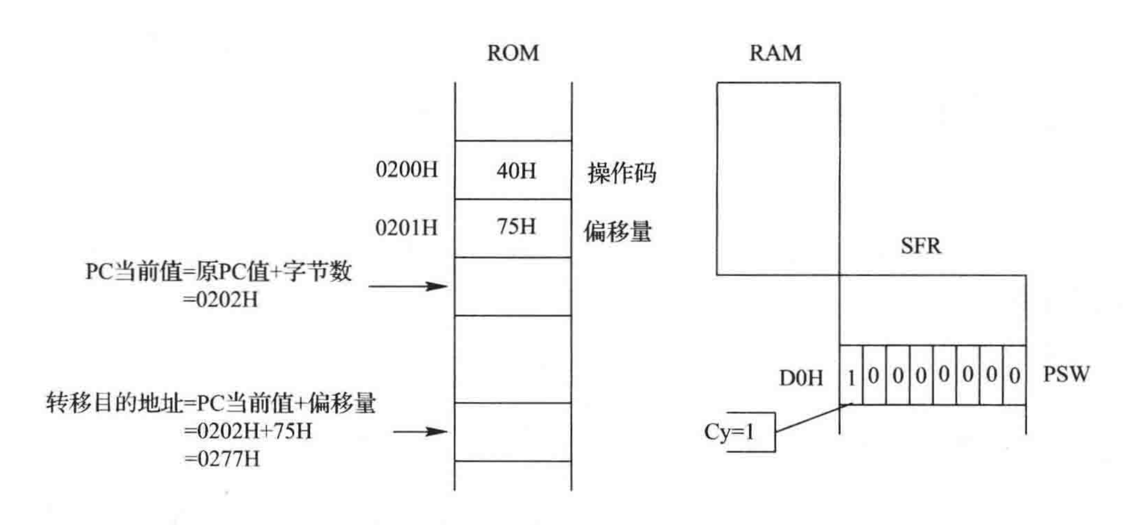


图3-6 指令"JC rel"的执行示意图

指令"JC rel"是一条双字节指令，在ROM中的编码(十六进制形式)为40H、75H。该指令的功能是检测进位标志位Cy(即PSW.7)，当Cy＝1时程序转移到PC当前值加上偏移量rel所指示的目的地址开始执行，否则Cy＝0，程序不转移，顺序往下执行。已知Cy＝1，rel＝75H，且指令存放在ROM的0200H、0201H单元中。CPU取出该双字节指令后，PC的当前值为0202H，该值与偏移量75H相加得到目的地址为0277H，即程序将转向0277H单元取指令并执行，跳过了0203H～0276H单元的程序段。

采用汇编语言编程时，常用一个自定义的标号(即符号地址)来表示相对地址偏移量rel，程序汇编时自动计算偏移量。

3.2.7 位寻址

位寻址是对片内RAM的位寻址区和某些可位寻址的SFR进行位操作时的寻址方式。

【例3-7】 如果位地址20H的内容为1，则指令"MOV C，20H"执行后，位地址PSW.7的内容为1，执行过程如图3-7所示。

指令"MOV C，20H"在ROM中的编码(十六进制形式)为A2H、20H。指令执行时，CPU在对操作码进行分析后得知该指令的功能是将位地址20H单元(对应片内RAM中24H单元的最低位)的内容复制到位累加器C(对应PSW的最高位)中，C中原来的内容

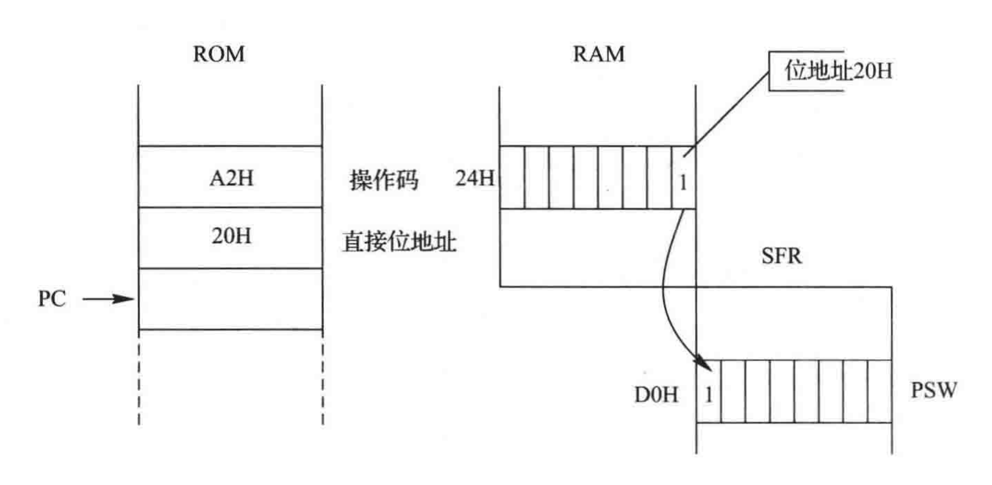


图 3-7　指令“MOV　C，20H”的执行示意图

被覆盖。因为位地址 20H 单元的内容为“1”，所以指令执行后位累加器 C(即 PSW.7)的内容变为“1”。

位寻址的寻址范围如下：

(1) 片内 RAM 低 128 字节中位寻址区的 128 位(见表 2-3)，该区域的可寻址位有两种表示方法：

- 直接使用位地址表示；
- 使用单元地址加位序号表示。

例如，位地址 00H 和 20H.0 指的都是片内 RAM 中 20H 单元的第 0 位。编程中常用“位地址”表示方法。

(2) SFR 的可寻址位，可供位寻址的特殊功能寄存器有 11 个，地址的尾数是 0 或 8 的寄存器(见表 2-4)可以位寻址。该区域的可寻址位有 4 种表示方法：

- 使用位名称表示；
- 直接使用位地址表示；
- 使用单元地址加位序号表示；
- 使用 SFR 符号加位序号表示。

例如，中断允许寄存器 IE 如下：

	D7	D6	D5	D4	D3	D2	D1	D0
(A8H)	AFH	AEH	ADH	ACH	ABH	AAH	A9H	A8H
IE	EA			ES	ET1	EX1	ET0	EX0

其中，最高位是中断允许总控制位。位名称是 EA，直接位地址是 0AFH，单元地址加位序号是 0A8H.7，SFR 符号加位序号是 IE.7。编程中常用“位名称”表示方法。

3.3　单片机的指令系统

按指令的功能不同，MCS-51 单片机指令系统可分为五大类：

(1) 数据传送类指令：实现存储器赋值、数据转移等功能。

(2) 算术运算类指令：实现数值的加、减、乘、除等运算功能。

(3) 逻辑运算类指令：实现逻辑与、或、异或、移位等功能。

(4) 控制转移类指令：实现程序条件转移、无条件转移等功能。

(5) 位操作类指令：实现位清0、置1、判断等功能，由MCS-51单片机内部特有的布尔处理器完成。

3.3.1　数据传送类指令

在MCS-51单片机中，数据传送是最基本和最主要的操作。数据传送操作可以在片内RAM单元和SFR中进行，也可以在累加器A和片外RAM之间进行，还可以到ROM中进行查表。在数据传送类指令中，除了以累加器A为目的操作数的指令会对PSW中的奇偶标志位P有影响外，其余指令执行时均不会影响任何标志位。

数据传送类指令可分为数据传送指令、数据交换指令和堆栈操作指令三类。

1. 数据传送指令

根据访问对象的不同，数据传送指令可进一步分为三种：访问片内RAM的MOV指令、访问片外RAM的MOVX指令和访问ROM的MOVC指令。

1) 访问片内RAM的MOV指令

数据传送指令是将数据进行单向传送，即将操作数从源操作数单元传送到目的操作数单元，指令执行后，源操作数不变，目的操作数修改为源操作数。该类指令用于实现数据在片内RAM单元之间、寄存器之间、寄存器与片内RAM单元之间的传送。该类指令采用的指令助记符为MOV，通用格式如下：

```
MOV  <目的操作数>，<源操作数>
```

其功能是将源操作数的内容复制到目的操作数所在单元，而源操作数的内容不变。在MCS-51单片机中，能够作源操作数的有A、Rn、direct、@Ri和#data(或#data16)，能够作目的操作数的有A、Rn、direct、@Ri和DPTR。将目的操作数和源操作数按照不同寻址方式进行组合就可派生出该类指令的全部形式。但是在源操作数和目的操作数进行组合时要遵守以下规则：

· 源操作数与目的操作数的类型要匹配；
· 除direct以外，源操作数与目的操作数不能相同；
· Rn和@Ri之间不能相互传送。

基于上述规则，分别以A、Rn、direct、@Ri和DPTR作目的操作数，可以构造出如下五组指令。

(1) 以A为目的操作数的指令：

```
MOV    A, Rn        ; (Rn)→(A)
MOV    A, direct    ; (direct)→(A)
MOV    A, @Ri       ; ((Ri))→(A)
MOV    A, #data     ; data →(A)
```

其功能是将源操作数的内容传送到累加器A中，源操作数的寻址方式分别为寄存器寻址、

直接寻址、寄存器间接寻址和立即寻址。

（2）以 Rn 为目的操作数的指令：

```
MOV    Rn，A              ；(A)→(Rn)
MOV    Rn，direct         ；(direct)→(Rn)
MOV    Rn，#data          ；data →(Rn)
```

其功能是将源操作数的内容传送到工作寄存器 Rn 中，源操作数的寻址方式分别为寄存器寻址、直接寻址和立即寻址。这里的 Rn 可以是工作寄存器 R0～R7 中的某一个，因此上面一条基本指令可以变成 8 条具体指令。

（3）以 direct 为目的操作数的指令：

```
MOV    direct，A                ；(A)→(direct)
MOV    direct，Rn               ；(Rn)→(direct)
MOV    direct1，direct2         ；(direct2)→(direct1)
MOV    direct，@Ri              ；((Ri))→(direct)
MOV    direct，#data            ；data →(direct)
```

其功能是将源操作数的内容送入由直接地址指出的片内 RAM 单元，源操作数的寻址方式分别为寄存器寻址、直接寻址、寄存器间接寻址和立即寻址。

（4）以@Ri 为目的操作数的指令：

```
MOV    @Ri，A                   ；(A)→((Ri))
MOV    @Ri，direct              ；(direct)→((Ri))
MOV    @Ri，#data               ；data →((Ri))
```

其功能是将源操作数的内容传送到 Ri 内容所指向的地址单元中，源操作数的寻址方式分别为寄存器寻址、直接寻址和立即寻址。这里的 Ri 只能是工作寄存器 R0 或 R1。

（5）以 DPTR 为目的操作数的指令：

```
MOV    DPTR，#data16              ；data →(DPTR)
```

这是 MCS－51 单片机指令系统中唯一的一条 16 位数据传送指令，该指令的功能是将 16 位立即数传送到数据指针 DPTR 中。由于 DPTR 是由 DPH 和 DPL 组成的，因此这条指令执行后要把 data16 的高 8 位数据传送给 DPH，而把低 8 位数据传送给 DPL。例如，指令“MOV DPTR，#1234H”执行后，DPH 中的值为“12H”，DPL 中的值为“34H”。为实现该功能，我们也可以分别向 DPH 和 DPL 送数，即用两条指令“MOV DPH，#12H”和“MOV DPL，#34H”实现上述功能。其中 DPH 和 DPL 是特殊功能寄存器，属于 direct 类型。

【例 3－8】 试用不同方法实现以下传送功能：

① 把存放在片内 RAM 50H 单元中的数据 20H 传送到累加器 A 中(要求最终指令以 A 为目的操作数)。

```
程序 1:  MOV   A, 50H       ; (A)=(50H)= 20H
程序 2:  MOV   R0, 50H      ; (R0)=(50H)= 20H
         MOV   A, R0        ; (A)=(R0)= 20H
程序 3:  MOV   R0, #50H     ; (R0)= 50H
         MOV   A, @R0       ; (A)=((R0))=(50H)= 20H
```

② 把存放在片内 RAM 20H 单元中的数据 55H 送到寄存器 R7 中(要求最终指令以 R7 为目的操作数)。

```
程序 1:  MOV   R7, 20H      ; (R7)=(20H)= 55H
程序 2:  MOV   A, 20H       ; (A)=(20H)= 55H
         MOV   R7, A        ; (R7)=(A)=(20H)= 55H
```

③ 把存放在片内 RAM 30H 单元中的数据 00H 送到 60H 单元中(要求最终指令以 60H 为目的操作数)。

```
程序 1:  MOV   60H, 30H     ; (60H)=(30H)= 00H
程序 2:  MOV   R2, 30H      ; (R2)=(30H)= 00H
         MOV   60H, R2      ; (60H)=(R2)=(30H)= 00H
程序 3:  MOV   A, 30H       ; (A)=(30H)= 00H
         MOV   60H, A       ; (60H)=(A)=(30H)= 00H
```

④ 把存放在片内 RAM 30H 单元中的数据 00H 送到 60H 单元中(要求最终指令以 @Ri为目的操作数)。

```
程序 1:  MOV   R1, #60H     ; (R1)= 60H
         MOV   @R1, 30H     ; ((R1))=(60H)=(30H)= 00H
程序 2:  MOV   R1, #60H     ; (R1)= 60H
         MOV   A, 30H       ; (A)=(30H)= 00H
         MOV   @R1, A       ; ((R1))=(60H)=(A)=(30H)= 00H
```

【例 3-9】 设(30H)= 40H, (40H)= 10H, (10H)= 00H, (P1)=11001010B, 试分析如下程序段中的指令分别属于上述 16 条访问片内 RAM 的数据传送指令中的哪一条? 指令执行后各单元中的内容是什么?

```
MOV   R0, #30H
MOV   A, @R0
MOV   R1, A
MOV   B, @R1
MOV   @R1, P1
MOV   P2, P1
MOV   10H, #20H
```

解: “MOV　R0, #30H”属于“MOV　Rn, #data”格式, 执行后(R0)=30H。

“MOV　A, @R0”属于“MOV　A, @Ri”格式, 执行后(A)=((R0))=(30H) =40H。

“MOV　R1，A”属于“MOV　Rn，A”格式，执行后(R1)=(A)=40H。

“MOV　B，@R1”属于“MOV　direct，@Ri”格式，执行后(B)=((R1))=(40H)=10H。

“MOV　@R1，P1”属于“MOV @Ri，direct”格式，执行后((R1))=(40H)=(P1)=0CAH

“MOV　P2，P1”属于“MOV　direct1，direct2”格式，执行后(P2)=(P1)=0CAH。

“MOV　10H，#20H”属于“MOV　direct，#data”格式，执行后(10H)= 20H。

所有指令执行后，(10H)=20H，(30H)=40H，(40H)=(P2)=(P1)=0CAH。

2）访问片外 RAM 的 MOVX 指令

在 MCS-51 指令系统中，CPU 对片外 RAM 进行数据传送，必须通过累加器 A 且采用寄存器间接寻址的方法来完成，该类指令采用的指令助记符为 MOVX，有以下单字节指令：

```
MOVX    A，@DPTR        ；((DPTR))→(A)
MOVX    A，@Ri          ；((Ri))→(A)
MOVX    @DPTR，A        ；(A)→((DPTR))
MOVX    @Ri，A          ；(A)→((Ri))
```

前两条指令的功能是把片外 RAM 中的一个字节读到累加器 A 中，后两条指令的功能是把累加器 A 中一个字节的数据写到片外 RAM 中。若采用 DPTR 间接寻址，高 8 位地址(由 DPH 提供)由 P2 口输出，低 8 位地址(由 DPL 提供)由 P0 口输出，可寻址片外 RAM 的全部 64KB 单元。若采用 Ri(i=0，1)间接寻址，低 8 位地址(由 Ri 提供)由 P0 口输出，能寻址片外 RAM 的 256 个单元。因为高 8 位地址由 P2 口提供，如果对 P2 口进行相应的修改，也可寻址片外 RAM 的全部 64KB 单元。

【例 3-10】 试写出完成以下功能需要的指令序列：

① 将片内 RAM 30H 单元中的内容送入片外 RAM 1000H 单元中；

② 将片外 RAM 1000H 单元中的内容送入片内 RAM 40H 单元中；

③ 将片外 RAM 1000H 单元中的内容送入片外 RAM 2000H 单元中。

参考程序：

```
① MOV    A，30H          ；将片内 RAM 30H 单元中的内容送入 A 中
  MOV    DPTR，#1000H    ；将片外 RAM 的地址 1000H 送入 DPTR
  MOVX   @DPTR，A        ；将 A 中的内容写到 DPTR 提供的地址中
② MOV    DPTR，#1000H    ；将片外 RAM 的地址 1000H 送入 DPTR
  MOVX   A，@DPTR        ；将 DPTR 提供的地址中的内容读到 A 中
  MOV    40H，A          ；将 A 中的内容送入片内 RAM 40H 单元中
③ MOV    DPTR，#1000H    ；将片外 RAM 的地址 1000H 送入 DPTR
  MOVX   A，@DPTR        ；将 DPTR 提供的地址中的内容读到 A 中
  MOV    DPTR，#2000H    ；将片外 RAM 的地址 2000H 送入 DPTR
  MOVX   @DPTR，A        ；将 A 中的内容写到 DPTR 提供的地址中
```

因为在 MCS-51 单片机中，与片外 RAM 打交道的只能是累加器 A，所以所有需要送

入片外 RAM 的数据必须通过累加器 A 送出去，而所有要读入的片外 RAM 中的数据也必须先读入到累加器 A 中。并且使用时应先将要读或写的地址送入 DPTR 中，然后再用读/写命令。

注意：片外扩展的 I/O 接口也要利用这 4 条指令进行数据的输入/输出。

3）访问 ROM 的 MOVC 指令

通常 ROM 中可以存放两类内容：一是单片机执行的程序代码；二是一些固定不变的常数(如表格数据、字符代码等)。访问 ROM 实际是指读取 ROM 常数表中的数据，简称查表，而访问 ROM 的数据传送指令被称为查表指令。该指令必须通过累加器 A 且采用变址寻址的方法来完成，采用的指令助记符为 MOVC，有以下单字节指令：

```
MOVC    A, @A+DPTR        ; (A)+((DPTR))→(A)
MOVC    A, @A+PC          ; (PC)+1→(PC), ((A)+(PC))→(A)
```

第一条指令以数据指针 DPTR 作为基址寄存器，累加器 A 的内容作为无符号数和 DPTR 的内容相加得到一个 16 位的地址，把该地址指出的 ROM 单元的内容送到累加器 A 中。该指令的执行结果只和指针 DPTR 及累加器 A 的内容有关，与该指令存放的地址及常数表格存放的地址无关，因此表格的大小和位置可以在 64KB 的 ROM 中任意安排，且一个表格可以为各个程序块公用，所以该指令称为远程查表指令。

第二条指令以程序计数器 PC 作为基址寄存器，累加器 A 的内容作为无符号数和 PC 的当前值(下一条指令的起始地址)相加后得到一个 16 位的地址，把该地址指出的 ROM 单元的内容送到累加器 A 中。该指令只能在查表指令后的 256B 的地址空间寻址，因此表格的大小受到限制，且表格只能被一段程序所利用，所以该指令称为近程查表指令。该指令的优点是不改变特殊功能寄存器 DPTR 的状态，只要根据 A 的内容就可以取出表格中的常数。

【例 3-11】 假设从 ROM 的 1000H 单元开始存放 0～9 的平方值，如果以 DPTR 为基址寄存器进行查表得 5 的平方值，其程序段如下：

```
MOV     DPTR, #1000H
MOV     A, #05H
MOVC    A, @A+DPTR
```

执行结果为(DPTR)＝1000H，(A)＝19H。

如果以 PC 作为基址寄存器进行上述查表功能，并假设 MOVC 指令所在地址(PC)＝0FF5H，则在执行查表指令之前要进行地址调整，因为 PC 的当前值与表格首地址的偏移量为 1000H－(0FF5H＋1)＝0AH，所以相应的程序段如下：

```
MOV     A, #05H
ADD     A, #0AH           ; 用加法指令进行地址调整
MOVC    A, @A+PC
```

执行结果为(PC)＝0FF6H，(A)＝19H。

2. 数据交换指令

数据交换指令是将数据进行双向传送，是两个字节间或两个半字节间的双向交换，涉及传送的双方互为源操作数、目的操作数，指令执行后双方的操作数互换。因此，两操作数均未丢失。数据交换指令又分为字节交换指令和半字节交换指令两种。

1）字节交换指令

字节交换指令包括以下3条：

```
XCH    A, Rn        ; (A)←→(Rn)
XCH    A, direct    ; (A)←→(direct)
XCH    A, @Ri       ; (A)←→((Ri))
```

这3条指令的功能是将累加器A中的数据与源操作数中的数据进行互换。即指令执行后，源操作数中的内容与交换前的累加器A中的内容相同，累加器A中的内容与交换前的源操作数中的内容相同。

2）半字节交换指令

半字节交换指令如下：

```
XCHD    A, @Ri      ; (ACC.3～ACC.0)←→((Ri).3～(Ri).0)
SWAP    A           ; (ACC.3～ACC.0)←→(ACC.7～ACC.4)
```

第一条指令的功能是将累加器A的低4位(低半字节)与间接寄存器所指向的地址单元中的低4位互换，而各自的高4位(高半字节)保持不变。

第二条指令的功能是将累加器A内部的高4位与低4位的内容互换。

【例3-12】 已知(R0)= 30H，(30H)= 4AH，(A)= 28H，试分析下列指令执行的结果：

① XCH　A，@R0

② XCHD　A，@R0

③ SWAP　A

解：① 程序执行后，(A)= 4AH，(30H)= 28H。

② 程序执行后，(A)= 2AH，(30H)= 48H。

③ 程序执行后，(A)= 82H。

3. 堆栈操作指令

堆栈是按"后进先出"的规则组织的一片存储区域。在MCS-51单片机中，堆栈设置在片内RAM的低128字节单元，堆栈的栈顶由堆栈指针SP指出。

堆栈操作指令是一种特殊的数据传送指令，其特点是根据SP中栈顶地址进行数据传送操作。这类指令共有如下2条：

```
PUSH    direct    ; (SP)+1→(SP), (direct)→((SP))
POP     direct    ; ((SP))→(direct), (SP)-1→(SP)
```

第一条指令为入栈指令，其功能是把direct单元中的操作数传送到堆栈中去。这条指

令执行时分为两步：第一步，将 SP 中的栈顶地址加 1，使之指向堆栈的新的栈顶单元；第二步，把 direct 所指定的单元中的数据压入由 SP 所指示的栈顶单元。

第二条指令为出栈指令，其功能是把堆栈中的操作数传送到 direct 单元。这条指令执行时也分为两步：第一步，把由 SP 所指示的栈顶单元中的操作数传送到 direct 单元；第二步，将 SP 中的原栈顶地址减 1，使之指向新的栈顶地址。

入栈指令和出栈指令一般是成对出现的。

【例 3－13】 把数据指针 DPTR、状态标志寄存器 PSW、累加器 A 中的数据入栈保护。

解：执行结果如图 3－8 所示，完成该功能的程序如下：

```
PUSH    DPL    ；先令 SP 中的栈顶地址加 1，然后将 DPTR 的低 8 位送入(SP＋1)
PUSH    DPH    ；先令 SP 中的栈顶地址加 1，然后将 DPTR 的高 8 位送入(SP＋2)
PUSH    PSW    ；先令 SP 中的栈顶地址加 1，然后将 PSW 的内容送入(SP＋3)
PUSH    ACC    ；先令 SP 中的栈顶地址加 1，然后将 A 的内容送入(SP＋4)
```

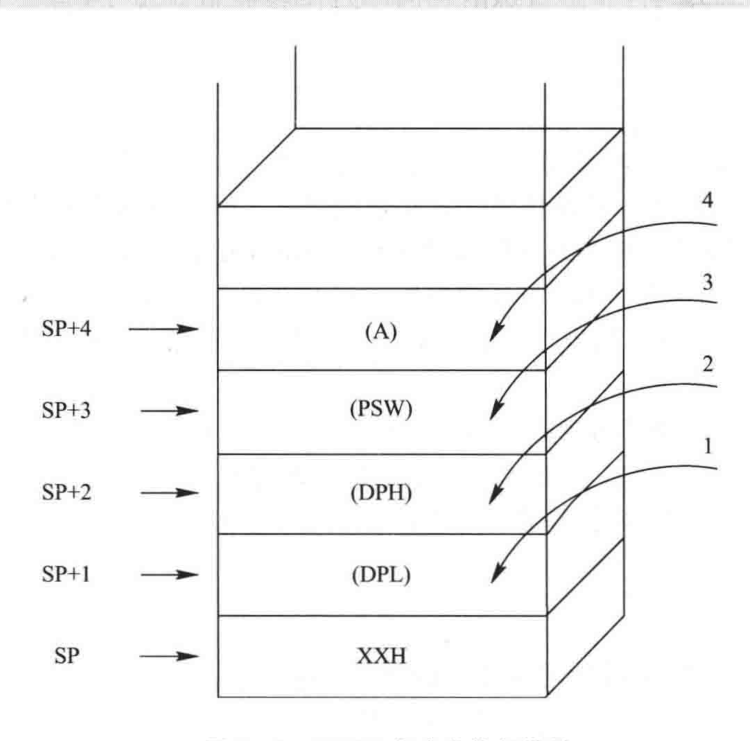


图 3－8　PUSH 指令的执行结果

【例 3－14】 将上例中在堆栈里保存的数据恢复(弹出)到 DPTR、PSW、A 中。

解：执行结果如图 3－9 所示，完成该功能的程序如下：

```
POP     ACC     ；先将(SP)的内容送到 A 中，然后令 SP 中的栈顶地址减 1
POP     PSW     ；先将(SP－1)的内容送到 PSW 中，然后令 SP 中的栈顶地址减 1
POP     DPH     ；先将(SP－2)的内容送到 DPH 中，然后令 SP 中的栈顶地址减 1
POP     DPL     ；先将(SP－3)的内容送到 DPL 中，然后令 SP 中的栈顶地址减 1
```

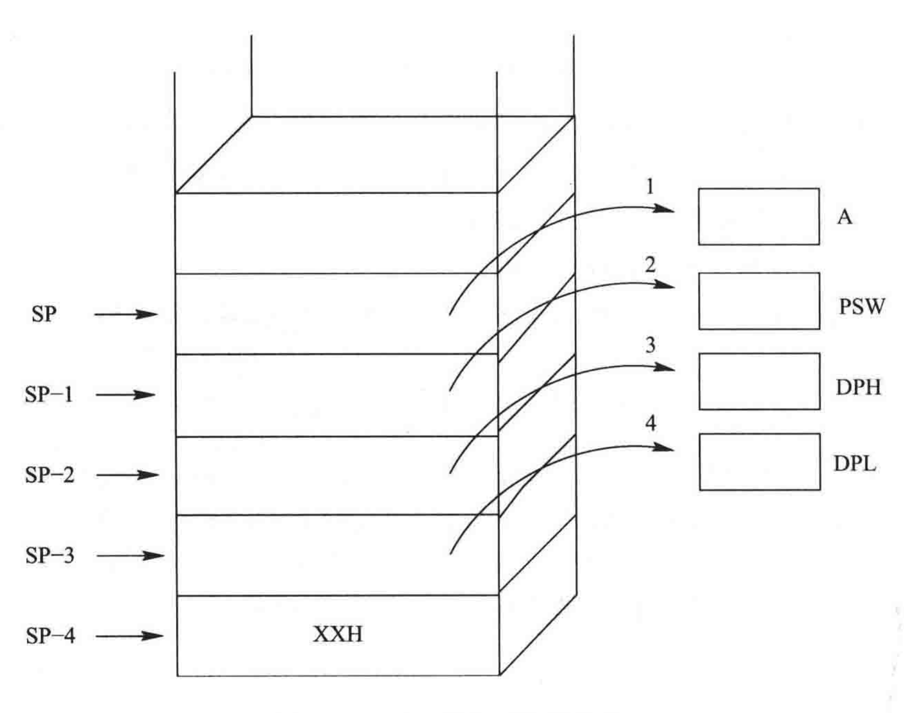


图 3-9 POP 指令的执行结果

这两个例子说明入栈和出栈的数据只有按照“后进先出”的原则，才能实现恢复数据的功能。另外，堆栈操作指令的操作数是直接地址单元，所以用 ACC(累加器 A 的符号地址)来表示累加器 A，即堆栈操作中的累加器入栈和出栈必须对 ACC 操作。特殊功能寄存器 SFR 可以直接使用寄存器名进行堆栈操作，如“PUSH DPH”。工作寄存器 R0～R7 不可以直接用 PUSH 和 POP 指令进行堆栈操作，需用直接地址 00H～07H(第 0 组)，如 R0 入栈，需用“PUSH 00H”。

堆栈指令多用在子程序或中断服务程序中，保护以前的某些重要数据，即保护现场。此外，还可以实现片内 RAM 单元之间的数据传送和交换。

【例 3-15】 用堆栈指令实现 RAM 10H 和 20H 中的内容交换，设(10H)=12H，(20H)=34H。

参考程序：

```
MOV     SP, #6FH      ; 设置堆栈指针指向 6FH
PUSH    10H           ; (SP)+ 1→(SP), (SP)=70H, (70H)=12H
PUSH    20H           ; (SP)+ 1→(SP), (SP)=71H, (71H)=34H
POP     10H           ; (10H)=34H, (SP)-1→(SP), (SP)=70H
POP     20H           ; (20H)=12H, (SP)-1→(SP), (SP)=6FH
```

3.3.2 算术运算类指令

算术运算类指令可以分为加法、减法、乘法和除法指令 4 类，这一类指令的最大特点是执行后可能会影响状态标志寄存器 PSW 的某些标志位。

1. 加法指令

加法指令可分为 4 类：不带 Cy 的加法指令、带 Cy 的加法指令、加 1 指令和十进制调

整指令。

1）不带 Cy 的加法指令

不带 Cy 的加法指令有以下 4 条：

```
ADD     A, Rn        ; (A)+(Rn)→(A), n = 0～7
ADD     A, direct    ; (A)+(direct)→(A)
ADD     A, @Ri       ; (A)+((Ri))→(A), i = 0 或 1
ADD     A, #data     ; (A)+data→(A)
```

这 4 条加法指令的一个加数必须由累加器 A 提供，而另一个加数可由寄存器寻址、直接寻址、寄存器间接寻址和立即寻址 4 种不同的寻址方式得到，相加的结果再存入累加器 A 中。这些指令执行后，根据指令的执行情况，硬件自动对 PSW 中的 Cy、Ac、OV 和 P 四个标志位重新进行置 1 或清 0：

- 加法运算结果中，有向最高位进位时，Cy=1，反之 Cy=0；
- 加法运算结果中，有低 4 位向高 4 位进位时，Ac=1，反之 Ac=0；
- 加法运算结果中，第 6 位和第 7 位不同时产生进位时，则代表运算有溢出，即 OV=1，反之 OV=0；
- 累加器 A 中有奇数个“1”时，P=1，反之 P=0。

【例 3-16】 分析如下程序段执行后，累加器 A 及 PSW 相关标志的结果。

```
MOV     A, #10010010B
ADD     A, #11001001B
```

解：第一条指令将立即数 92H 送入 A 中，第二条指令将立即数 C9H 与 A 中的 92H 相加，结果存入 A 中。

指令执行后，(A)=01011011B =5BH，Cy=1，Ac=0，OV=1，P=1。

2）带 Cy 的加法指令

带 Cy 的加法指令有以下 4 条：

```
ADDC    A, Rn        ; (A)+(Rn)+Cy→(A), n=0～7
ADDC    A, direct    ; (A)+(direct)+Cy→(A)
ADDC    A, @Ri       ; (A)+((Ri))+Cy→(A), i=0, 1
ADDC    A, #data     ; (A)+data+Cy→(A)
```

这组指令与不带 Cy 的加法指令的区别在于：带 Cy 的加法指令在执行加法运算时必须考虑该指令执行之前已经存在的进位标志位 Cy，即这组指令有三个操作数参加加法运算。

与不带 Cy 的加法指令的相同之处在于：指令执行后，结果存入累加器 A 中，并根据指令的执行情况，重新对 Cy、Ac、OV、P 等标志位置 1 或清 0。

带 Cy 的加法指令常用于多字节数加法运算中。

【例 3-17】 已知当前的 Cy=1，分析下列指令的执行结果。

```
MOV     A, #85H
ADDC    A, #97H
```

解：执行结果为(A)＝1DH，Cy＝1，Ac＝0，OV＝1，P＝0。

【例 3－18】 若16位二进制数分别存放在工作寄存器R1 R0和R3 R2中，试编程计算R1 R0＋ R3 R2，并将结果存放在R5 R4中。

分析：因为MCS－51单片机的CPU是8位的，所以只能进行单字节加法运算，若想实现多字节加法运算，比如本例的R1 R0＋R3 R2，首先应进行两个低8位字节的加法运算，此时不需要考虑加法指令执行之前存在的进位标志位Cy，即只需要计算R0＋R2。接下来要进行高位字节加法运算时必须考虑前一步加法运算产生的进位标志位Cy，即需要计算R1＋R3＋Cy。注意，加法指令的一个加数必须由累加器A提供。

参考程序：

```
MOV     A, R0
ADD     A, R2       ; (R0)+(R2)→(A)和 Cy
MOV     R4, A
MOV     A, R1
ADDC    A, R3       ; (R1)+(R3)+Cy→(A)和 Cy
MOV     R5, A
```

3）加1指令

加1指令包括以下5条：

```
INC     A       ; (A)+1→(A)，影响 P 标志
INC     Rn      ; (Rn)+1→(Rn)
INC     direct  ; (direct)+1→(direct)
INC     DPTR    ; (DPTR)+1→(DPTR)
INC     @Ri     ; ((Ri))+1→((Ri))
```

这组指令都是单操作数指令，指令中的操作数既为源操作数又为目的操作数。这组指令的功能是把源操作数所指定的单元内容加1，结果再送回原来单元。除"INC　A"指令可能会影响P标志位外，其余指令不影响PSW的标志位。

Ri和DPTR常用作指针并指向一片存储区域的起始地址，将它们的内容加1可以指向下一单元的地址，编程时常利用这两个指针对一片存储区域进行操作。

【例 3－19】 将片内RAM中30H为起始地址的3个无符号数相加，并将结果(假设小于256)存放到40H单元中，试编程实现该功能。

分析：要想实现该程序的功能，可以采取多种方法，比如直接从存储区提取数据进行顺序相加或利用指针从存储区提取数据进行顺序相加。

参考程序1：

```
MOV     A, 30H
ADD     A, 31H
ADD     A, 32H
MOV     40H, A
```

参考程序 2：

```
MOV    A, #00H        ；将A清0
MOV    R0, #30H       ；将地址30H送入R0中
ADD    A, @R0
INC    R0             ；将R0的内容加1，变成31H
ADD    A, @R0
INC    R0             ；将R0的内容加1，变成32H
ADD    A, @R0
MOV    40H, A
```

因为本例涉及的存储单元个数很少，所以程序1更简单。

4）十进制调整指令

十进制调整指令如下：

```
DA    A
```

其功能是在进行BCD码加法运算时，跟在ADD和ADDC指令之后，对相加后存放在累加器A中的运算结果进行十进制调整：若累加器低4位大于9或辅助进位位Ac=1，则将累加器作加6调整，即低4位进行加6修正；若累加器高4位大于9或进位位Cy=1，则将累加器作加60H调整，即高4位进行加6修正。

因为指令是对BCD码进行修正，所以该指令也称为BCD码修正指令。两个压缩BCD码按二进制数相加后，必须经本指令的调整才能得到正确的结果。

【例3-20】 试编写程序，实现95+59的BCD码加法。

参考程序：

```
MOV    A, #95H
ADD    A, #59H      ；(A)=EEH
DA     A            ；(A)=54H
```

上述程序段执行后，(A)=54H，Cy=1，即95+59=154，结果是正确的。若不采用“DA　A”进行调整，结果为EEH就不正确了。

2. 减法指令

减法指令可分为带Cy的减法指令和减1指令两种。

1）带Cy的减法指令

带Cy的减法指令包括以下4条：

```
SUBB    A, Rn        ；(A)-Cy-(Rn)→(A)，n=0～7
SUBB    A, direct    ；(A)-Cy-(direct)→(A)
SUBB    A, @Ri       ；(A)-Cy-((Ri))→(A)，i=0或1
SUBB    A, #data     ；(A)-Cy-data→(A)
```

其功能是用累加器A提供的操作数减去指令执行前的Cy值和源操作数所指内容，最终结果存在A中。这些指令执行后，根据指令的执行情况，硬件重新对PSW中的Cy、Ac、OV和P四个标志位进行置1或清0：

- 当减法运算结果的最高位有借位时，Cy=1，否则 Cy=0；
- 当减法运算的低 4 位向高 4 位有借位时，Ac=1，否则 Ac=0；
- 在减法运算中，第 6 位和第 7 位不同时产生借位时，OV=1，否则 OV=0；
- 当累加器 A 中有奇数个“1”时，P=1，否则 P=0。

MCS-51 单片机指令系统中没有不带 Cy 的减法指令，若用此组指令完成不带 Cy 的减法功能，必须将 Cy 先清 0。带 Cy 的减法指令常用于多字节数减法运算中。

【例 3-21】 已知 Cy=1，分析下列指令的执行结果。

```
MOV     A，#79H
SUBB    A，#56H
```

解： 执行结果为(A)=22H，Cy=0，Ac=0，OV=0，P=0。

【例 3-22】 试编写 1234H－0FA3H 的程序段，将结果高 8 位存入 51H，低 8 位存入 50H 单元。

分析： 与多字节加法运算一样，MCS-51 单片机若想实现多字节减法运算，首先应进行两个低 8 位字节的减法运算，此时不需要考虑之前存在的借位标志位 Cy。因为 MCS-51单片机中没有不带 Cy 的减法指令，所以在利用 SUBB 指令执行该步减法操作之前需要将 Cy 先清 0。接下来进行高位字节减法运算时直接利用 SUBB 指令即可，因为该指令考虑了前一步减法运算产生的借位标志位 Cy。注意，减法指令的一个减数必须由累加器 A 提供。

参考程序：

```
CLR     C           ；清进位位
MOV     A，#34H     ；将被减数低 8 位放入 A
SUBB    A，#0A3H    ；减法运算
MOV     50H，A      ；存低 8 位结果
MOV     A，#12H     ；将被减数高 8 位放入 A
SUBB    A，#0FH     ；减法运算
MOV     51H，A      ；存高 8 位结果
```

2）减 1 指令

减 1 指令包括以下 4 条：

```
DEC     A        ；(A)-1→(A)
DEC     Rn       ；(Rn)-1→(Rn)，n=0～7
DEC     direct   ；(direct)-1→(direct)
DEC     @Ri      ；((Ri))-1→((Ri))，i=0，1
```

其功能是把源操作数所指定的单元内容减 1，结果再送回原来单元。除“DEC　A”指令可能会影响 P 标志位外，其余指令不影响 PSW 的标志位。

3. 乘法指令

乘法指令如下：

```
MUL     AB       ；((A)×(B))15~8→(B)，((A)×(B))7~0→(A)，0→Cy
```

其功能是将累加器A和寄存器B中的两个8位无符号数相乘，乘积为16位，高8位存放在B中，低8位存放在A中。运算结果影响PSW标志中的Cy、OV和P：指令执行后，Cy=0；若乘积大于FFH(255)，则OV=1，否则OV=0；奇偶校验位P由累加器A中"1"的奇偶性决定。

注意：在MCS-51单片机的指令系统中，乘法和除法指令的目的操作数和源操作数必须是A和B，且目的操作数和源操作数之间没有"，"分割符。

【例3-23】 分析下列指令执行的结果。

```
MOV    A, #4EH
MOV    B, #5DH
MUL    AB
```

解：执行结果为(A)=56H，(B)=1CH，Cy=0，OV=1，P=0。

4. 除法指令

除法指令如下：

```
DIV    AB    ;(A)÷(B)之商→(A)，(A)÷(B)之余数→(B)，0→Cy
```

其功能是将累加器A中的8位无符号整数除以寄存器B中的8位无符号整数，所得商存放在A中，余数存放在B中。运算结果影响PSW标志中的Cy、OV和P：指令执行后，Cy=0；若除数为0，则OV=1，否则OV=0；奇偶校验位P由累加器A中"1"的奇偶性决定。

【例3-24】 分析下列指令执行的结果。

```
MOV    A, #87H
MOV    B, #0CH
DIV    AB
```

解：执行结果为(A)=0BH，(B)=03H，Cy=0，OV=0，P=1。

3.3.3 逻辑运算类指令

逻辑运算类指令的功能是进行字节的逻辑与、或、异或、求反、清0、左右移位等操作。按照操作数数量不同可以分为单操作数逻辑运算指令和双操作数逻辑运算指令两种。

1. 单操作数逻辑运算指令

单操作数逻辑运算指令只能对累加器A的内容进行逻辑操作。

1）累加器A清零指令

累加器A清零指令如下：

```
CLR    A    ;0→(A)
```

其功能是将累加器A清0。指令执行后P=0，其他标志位不受影响。

2）累加器A取反指令

累加器A取反指令如下：

```
CPL    A    ;(/A)→(A)
```

其功能是将累加器 A 的 8 个位同时按位取反操作，不影响标志位。

3）累加器 A 循环移位指令

（1）累加器 A 循环左移指令：

```
RL    A    ；(An)→(An+1)，n=0～6，(A7)→(A0)
```

其功能是将累加器 A 中的内容循环左移 1 位。该指令不影响标志位，操作过程如图 3－10 所示。

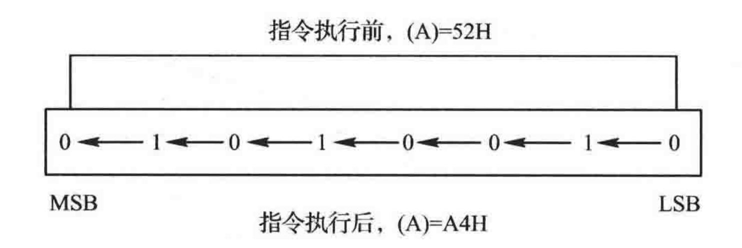


图 3－10 “RL A”指令的执行示意图

（2）累加器 A 连同 Cy 循环左移指令：

```
RLC    A    ；(An)→(An+1)，n=0～6，(A7)→Cy，Cy→(A0)
```

其功能是将累加器 A 的内容与 Cy 一起循环左移 1 位。该指令不影响 Ac 和 OV 标志位，但执行后将影响 Cy 和 P 标志位，操作过程如图 3－11 所示。

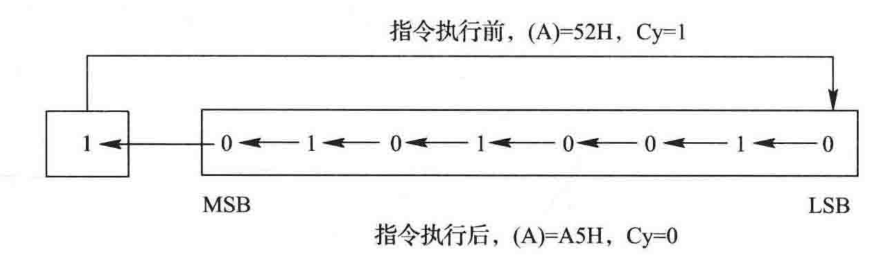


图 3－11 “RLC A”指令的执行示意图

（3）累加器 A 循环右移指令：

```
RR    A    ；(An+1)→(An)，n=0～6，(A0)→(A7)
```

其功能是将累加器 A 中的内容循环右移 1 位。该指令不影响标志位，操作过程如图 3－12 所示。

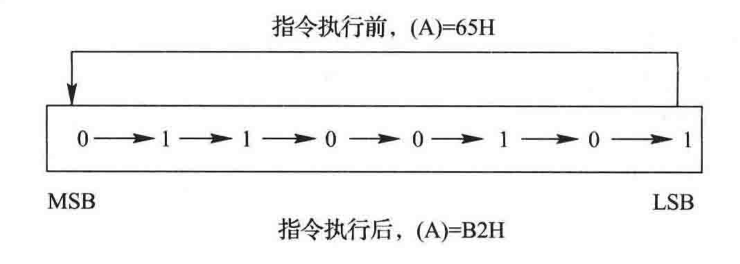


图 3－12 “RR A”指令的执行示意图

（4）累加器 A 连同 Cy 循环右移指令：

```
RRC    A    ；(An+1)→(An)，n=0～6，Cy→(A7)，(A0)→Cy
```

其功能是将累加器 A 的内容与 Cy 一起循环右移 1 位。指令执行后影响 Cy 和 P 标志位，

操作过程如图 3－13 所示。

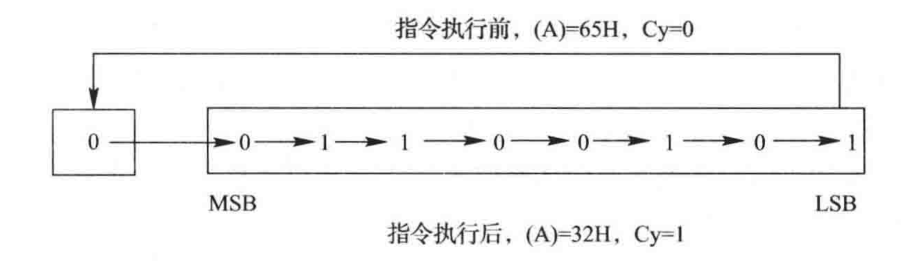


图 3－13　“RRC　A”指令的执行示意图

【例 3－25】　设(A)＝5AH(代表十进制数 90)，且 Cy＝0。

① 若执行指令“RLC　A”，结果为(A)＝B4H(代表十进制数 180)；

② 若执行指令“RRC　A”，结果为(A)＝2DH(代表十进制数 45)。

2. 双操作数逻辑运算指令

1) 逻辑与运算指令

逻辑与运算指令如下：

```
ANL    A, Rn             ; (A) ∧ (Rn)→(A)
ANL    A, direct         ; (A) ∧ (direct)→(A)
ANL    A, @Ri            ; (A) ∧ ((Ri))→(A)
ANL    A, #data          ; (A) ∧ data→(A)
ANL    direct, A         ; (direct) ∧ (A)→(direct)
ANL    direct, #data     ; (direct) ∧ data→(direct)
```

其功能是将源操作数单元的内容与目的操作数单元的内容按位相与，结果存放到目的操作数单元中，而源操作数单元中的内容不变。指令运行时仅影响 P 标志位。

逻辑与运算指令常用来屏蔽字节中的某些不用位，其他位保持不变。实现方法：欲屏蔽的位与“0”相与，保持不变的位与“1”相与。

【例 3－26】　已知(A)＝39H＝0011 1001B，试分析下列指令的执行结果。

① ANL　　A，#0FH

② ANL　　A，#0F0H

③ ANL　　A，#0FFH

解：① 因为 A 的高 4 位都与“0”相与，低 4 位都与“1”相与，所以高 4 位被屏蔽，低 4 位保持不变，即(A)＝09H＝0000 1001B。

② 因为 A 的高 4 位都与“1”相与，低 4 位都与“0”相与，所以高 4 位保持不变，低 4 位被屏蔽，即(A)＝30H＝0011 0000B。

③ 因为 A 的 8 位都与“1”相与，所以 8 位均保持不变，即(A)＝39H＝0011 1001B。

2) 逻辑或运算指令

逻辑或运算指令如下：

```
ORL    A, Rn             ; (A) ∨ (Rn)→(A)
ORL    A, direct         ; (A) ∨ (direct)→(A)
ORL    A, @Ri            ; (A) ∨ ((Ri))→(A)
ORL    A, #data          ; (A) ∨ data→(A)
```

```
ORL    direct, A          ; (direct)∨(A)→(direct)
ORL    direct, #data      ; (direct)∨ data→(direct)
```

其功能是将源操作数单元的内容与目的操作数单元的内容按位相或，结果存放到目的操作数单元中，而源操作数单元中的内容不变。指令运行时仅影响 P 标志位。

逻辑或运算指令常用来使字节中某些位置"1"，其他位保持不变。实现方法：欲置 1 的位与"1"相或，保持不变的位与"0"相或。

【例 3-27】 已知(A)＝39H＝0011 1001B，试分析下列指令的执行结果。

① ORL　　A，#0FH

② ORL　　A，#0F0H

③ ORL　　A，#00H

解：① 因为 A 的高 4 位都与"0"相或，低 4 位都与"1"相或，所以高 4 位保持不变，低 4 位被置 1，即(A)＝ 3FH＝0011 1111B。

② 因为 A 的高 4 位都与"1"相或，低 4 位都与"0"相或，所以高 4 位被置 1，低 4 位保持不变，即(A)＝0F9H＝1111 1001B。

③ 因为 A 的 8 位都与"0"相或，所以 8 位均保持不变，即(A)＝39H＝0011 1001B。

3）逻辑异或运算指令

逻辑异或运算指令如下：

```
XRL    A, Rn              ; (A)⊕(Rn)→(A)
XRL    A, direct          ; (A)⊕(direct)→(A)
XRL    A, @Ri             ; (A)⊕((Ri))→(A)
XRL    A, #data           ; (A)⊕ data→(A)
XRL    direct, A          ; (direct)⊕(A)→(direct)
XRL    direct, #data      ; (direct)⊕ data→(direct)
```

其功能是将源操作数单元的内容与目的操作数单元的内容相异或，结果存放到目的操作数单元中，而源操作数单元中的内容不变。指令运行时仅影响 P 标志位。

逻辑异或运算指令常用来使字节中某些位进行取反操作，其他位保持不变。实现方法：欲取反的位与"1"相异或，保持不变的位与"0"相异或。

【例 3-28】 已知(A)＝39H＝0011 1001B，试分析下列指令的执行结果。

① XRL　　A，#0FH

② XRL　　A，#0F0H

③ XRL　　A，#0FFH

解：① 因为 A 的高 4 位都与"0"相异或，低 4 位都与"1"相异或，所以高 4 位保持不变，低 4 位按位取反，即(A)＝ 36H＝0011 0110B。

② 因为 A 的高 4 位都与"1"相异或，低 4 位都与"0"相异或，所以高 4 位按位取反，低 4 位保持不变，即(A)＝0C9H＝1100 1001B。

③ 因为 A 的 8 位都与"1"相异或，故 8 位均按位取反，即(A)＝0C6H＝ 1100 0110B。

【例 3-29】 试编写程序，将存放在片外 RAM 2000H 单元中某数的低 4 位取反，高 2 位置 1，其余 2 位清 0。

分析：根据前面介绍可知，某些位欲清0应与“0”相与，保持不变的位应与“1”相与；某些位欲置1应与“1”相或，保持不变的位应与“0”相或；某些位欲取反应与“1”相异或，保持不变的位应与“0”相异或。因此，为实现片外RAM 2000H单元中某数的低4位取反，高2位置1，其余2位清0，需要先将该数从片外RAM读到累加器中，然后对该数的相应位进行“异或”、“或”及“与”操作，最后再将数据送回片外RAM。

参考程序：

```
MOV     DPTR, #2000H
MOVX    A, @DPTR
XRL     A, #00001111B          ；将2000H中的内容高4位保留，低4位取反
ORL     A, #11000000B          ；高2位置1
ANL     A, #11001111B          ；其余2位清0
MOVX    @DPTR, A
```

【例3-30】 试编写程序，将累加器A的低4位状态通过P1口的高4位输出，P1口的低4位状态不变。

参考程序：

```
ANL     A, #0FH            ；屏蔽A的高4位
SWAP    A                  ；将A的高、低4位交换
ANL     P1, #0FH           ；屏蔽P1口高4位
ORL     P1, A              ；利用P1口高4位输出A的低4位
```

3.3.4　控制转移类指令

控制转移类指令包括无条件转移指令、条件转移指令、子程序调用与返回指令、空操作指令4种。控制转移类指令的作用是根据要求改变PC值，以控制程序运行的方向，所有这些指令的目标地址都在ROM空间内。

1. 无条件转移指令

无条件转移是指当程序执行到该指令时，程序无条件转移到指令所提供的地址处执行。无条件转移指令包括长转移指令、短转移指令、相对转移指令和间接转移指令4条。

1）长转移指令

长转移指令如下：

```
LJMP    addr16      ；addr16→(PC)
```

这是一条三字节指令，提供了16位的转移目标地址，机器码为

02	$addr_{15\sim8}$	$addr_{7\sim0}$

执行LJMP指令时，将指令的第2字节和第3字节的内容分别装入PC的高8位和低8位中，使程序无条件转移到addr16指出的目标地址，不影响标志位。

由于指令中提供16位目标地址，所以执行这条指令可以使程序在全部64KB的ROM

空间内转移。

“LJMP　$”与“SSS：LJMP　SSS”相同。

2）短转移指令

短转移指令如下：

```
AJMP    addr11    ；(PC)+2→(PC)，addr11→(PC)10~0，(PC)15~11不变
```

这是一条双字节指令，提供了 11 位地址，机器码为

$addr_{10\sim8}$00001	$addr_{7\sim0}$

该指令的特点是由 addr11 的高 3 位与指令操作码 00001 共同组成指令的第一个字节，addr11的低 8 位组成指令的第二个字节。

本指令是为能与 MCS-48 的 JMP 指令兼容而设的。

3）相对转移指令

相对转移指令如下：

```
SJMP    rel    ；(PC)+2→(PC)，(PC)+rel→(PC)
```

这是一条双字节指令，第一个字节为操作码，第二个字节是偏移量 rel，其中 rel 是一个有符号的 8 位二进制数，以补码形式置于操作码之后存放，取值范围是－128～＋127。负数表示向地址减小的方向转移(反向转移)，正数表示向地址增加的方向转移(正向转移)，相对的基准是 PC 的当前值。因此，转移的目标地址为

```
目标地址＝SJMP 指令所在地址＋2＋rel
```

例如，在(PC)＝1000H 单元有一条“SJMP　rel”指令，若 rel＝3AH(＋58)，则程序正向转移到 1002H＋58＝1002H＋003AH＝103CH 地址处；若 rel＝F0H(－16)，则程序反向转移到 1002H－16＝1002H＋FFF0H＝0FF2H 地址处。

若 rel ＝ FEH(－2)，则目标地址＝(PC)＋2－2＝(PC)，即转移的目标地址就是“SJMP”指令的地址，在汇编指令中该偏移地址可用 $ 符号表示。若在程序的末尾加上“SJMP　$”指令(机器码为 80H、FEH)，则程序将在该处进行无限循环，进入等待状态。

用汇编语言编程时，指令中的偏移量 rel 常用目标地址的标号表示。汇编器能自动算出相对地址值。另外，**在汇编程序中长转移指令可以替代短转移指令和相对转移指令。**

4）间接转移指令

间接转移指令如下：

```
JMP    @A+DPTR    ；(A)+(DPTR)→(PC)
```

这是一条单字节指令，转移的目标地址由累加器 A 与数据指针 DPTR 的内容之和来确定，可以实现在 64 KB 范围内无条件转移。

该指令以 DPTR 的内容为基地址，A 的内容作变址，因此只要把 DPTR 的值固定，然后赋予 A 不同的值，即可实现程序的多分支转移。该指令在执行后不改变 DPTR 及 A 中原来的内容，也不影响标志位。**间接转移指令可代替众多的判别跳转指令，又称为散转指令。**

【例 3-31】 已知累加器 A 中存放着控制程序转向的编号 0～3，ROM 中存有起始地址

为TABLE的三字节长转移指令表，试编程使单片机能按照累加器A中的编号转去执行相应的命令程序，即当(A)=00H时，执行TAB0分支程序；当(A)=01H时，执行TAB1分支程序；当(A)=02H时，执行TAB2分支程序；当(A)=03H时，执行TAB3分支程序。

分析：因为表格中的LJMP是三字节指令，所以执行查表指令之前应将累加器A的内容乘以3，以形成正确偏移量。有多种方法能够实现将A乘以3的功能，例如，可以先给B赋值3，然后执行"MUL　AB"指令；还可以先将Cy清零，并将A的内容暂时传递给其他存储单元/寄存器，然后令A执行带Cy的循环左移指令得到2×A，最后执行"ADD　A，direct/Rn"指令得到3×A。本例采用后一种方法实现将A乘以3的功能。

参考程序：

```
        CLR    C
        MOV    B, A
        RLC    A                ；得到2×A
        ADD    A, B             ；得到3×A
        MOV    DPTR, #TABLE     ；将TABLE地址送入DPTR寄存器中
        JMP    @A+DPTR          ；程序转到地址为A+DPTR的地方去执行
TABLE:  LJMP   TAB0             ；当(A)=0时，执行TAB0分支程序
        LJMP   TAB1             ；当(A)=1时，执行TAB1分支程序
        LJMP   TAB2             ；当(A)=2时，执行TAB2分支程序
        LJMP   TAB3             ；当(A)=3时，执行TAB3分支程序
```

MCS-51单片机的4条无条件转移指令各具特点，下面对它们的特点进行简单的总结：

①"LJMP　addr16"是三字节的长转移指令，指令提供16位直接目标地址，程序可以在64 KB地址空间内转移。

②"AJMP　addr11"是双字节的短转移指令，指令提供11位目标地址，程序的转移目标地址是在包含PC当前值在内的同一个2 KB区域。

③"SJMP　rel"是双字节的相对转移指令，指令提供相对偏移量，程序是在PC当前值为起始地址的−128～+127B范围内转移。

以上3条指令都能使程序转移到一个固定的目标地址处，原则上说所有使用SJMP或AJMP的地方都可以用LJMP来替代。

④"JMP　@A+DPTR"是单字节间接转移指令，与前3条指令相比，虽然该指令的用途也是跳转，但是转到什么地方去不能由标号简单地决定。因为转移地址是由A和DPTR相加构成的，根据A的不同值就可以实现程序的多分支转移。

2. 条件转移指令

条件转移是当某种条件满足时，程序执行转移；条件不满足时，程序仍按原来顺序执行。转移的条件可以是上一条指令或更前一条指令的执行结果(常体现在标志位上)，也可以是条件转移指令本身包含的某种运算结果。由于该类指令采用相对寻址，因此程序可在以PC当前值为中心的−128～+127B范围内转移。条件转移指令可以分为累加器判0条件转移指令、比较不相等条件转移指令和减1条件转移指令3种。

1) 累加器判零条件转移指令

累加器判0条件转移指令如下：

```
JZ    rel    ；若(A)=0，则(PC)+2+rel→(PC)；若(A)≠0，则(PC)+2→(PC)
JNZ   rel    ；若(A)≠0，则(PC)+2+rel→(PC)；若(A)=0，则(PC)+2→(PC)
```

JZ 指令的功能是若累加器(A)=0 就转移，否则就继续往下执行；JNZ 指令与 JZ 指令的功能正好相反，若累加器(A)≠0 就转移，否则就顺序执行。累加器 A 的内容是否为 0，是由这条指令以前的其他指令执行的结果决定的，执行这条指令不作任何运算，也不影响标志位。

这两条指令都是双字节相对转移指令，rel 为相对地址偏移量，当各自的条件满足时，程序转移的目标地址为(PC)+2+rel。该指令的转移范围在 PC 当前值为起始地址的 −128～+127B范围内。

【例 3-32】 试编写程序，将片内 RAM 以 40H 为起始地址的数据块传送到片外 RAM 以 2000H 为起始地址的区域，遇 0 中止。

分析：要实现多个存储区的内容传送，通常采用 Ri 指针指向片内 RAM 首地址，采用 DPTR 指向片外 RAM 首地址。对于片内 RAM 存储区内容的传送，可以采用 A 或其他存储单元作中转，但是涉及对片外 RAM 存储区的访问，必须通过 A 作中转。

根据本题的要求，需要先将发送端起始地址 40H 送入 Ri(i=0，1)，然后将接收端起始地址 2000H 送入 DPTR，再通过@Ri 将发送端起始地址 40H 中的内容送 A，并对 A 的内容进行如下判断：若 A 的内容为 0，则程序跳转到结束语句，否则通过 A 将 40H 中的内容送入片外 RAM 的 2000H 中。然后分别令 Ri 和 DPTR 加 1，使它们分别指向下一存储单元，继续读取数据、判断、传送或结束的循环过程。

参考程序：

```
        MOV    R0，#40H
        MOV    DPTR，#2000H
LOOP：  MOV    A，@R0
        JZ     DONE          ；若 A 的内容为 0，则跳至 DONE，否则往下走
        MOVX   @DPTR，A
        INC    R0
        INC    DPTR
        LJMP   LOOP          ；程序上跳回 LOOP 处执行
DONE：  LJMP   DONE          ；程序原地循环
```

2）比较不相等条件转移指令

比较不相等条件转移指令如下：

```
CJNE   A，#data，rel     ；若(A)≠data，则(PC)+3+rel→(PC)
CJNE   A，direct，rel    ；若(A)≠(direct)，则(PC)+3+rel→(PC)
CJNE   Rn，#data，rel    ；若(Rn)≠data，则(PC)+3+rel→(PC)
CJNE   @Ri，#data，rel   ；若((Ri))≠data，则(PC)+3+rel→(PC)
```

其功能是对两个规定的操作数进行比较，并根据比较的结果来决定是否转移。若两个操作数相等，则程序顺序执行；若两个操作数不相等，则程序进行转移，因为这组指令都是三字节指令，所以转移的目标地址为(PC)+3+rel。

这组指令可以判断两个无符号数是否相等，如果两数不相等，还能用 Cy 来反映哪个

数大，哪个数小。因为指令执行过程中的比较操作实际上是减法操作，只是不保存两数之差，但要影响 Cy 标志位：如果目的操作数的内容大于源操作数的内容，则 Cy=0；如果目的操作数的内容小于源操作数的内容，则 Cy=1。因此在程序转移后利用标志位 Cy 作进一步的判断，可实现三分支转移。

【例 3-33】 试用含有 CJNE 的指令编写程序，将片内 RAM 以 40H 为起始地址的数据块传送到片外 RAM 以 2000H 为起始地址的区域，遇 0 中止。

分析：本例同例 3-32，采用 Ri 指针指向片内 RAM 首地址，采用 DPTR 指向片外 RAM 首地址。不同之处在于：例 3-32 先将片内存储区内容读取到累加器 A，然后对 A 是否为 0 进行判断；而本例在读取片内存储区内容前，先利用 CJNE 指令将该存储区的内容与 0 进行比较，如果片内存储区内容为 0，则直接结束循环，否则通过累加器 A 作为中转将片内 RAM 的内容送入片外 RAM，传送指令与例3-32一样。

参考程序：

```
        MOV     R0，#40H
        MOV     DPTR，#2000H
LOOP：  CJNE    @R0，#00H，LOOP1
        LJMP    $
LOOP1： MOV     A，@R0
        MOVX    @DPTR，A
        INC     R0
        INC     DPTR
        LJMP    LOOP
```

【例 3-34】 已知工作寄存器 R0 中存放着一个无符号数 X，试编写程序求出下式的函数值 Y，并存入工作寄存器 R1 中。

$$Y=\begin{cases}AAH & X>10H\\ 00H & X=10H\\ FFH & X<10H\end{cases}$$

分析：因为 CJNE 指令不仅可以判断两个操作数是否相等，还可以利用标志位 Cy 来反映哪个数大，哪个数小，所以可以利用该指令实现三分支转移。

本例的解题思路为：首先将存放在 R0 的 X 传送到累加器 A，然后利用 CJNE 指令让 A 与 10H 进行比较，如果(R0)=10H，则令(R1)=00H；如果(R0)≠10H，则需要根据 Cy 来判断。如果 Cy=1，说明(R0)<10H，则令(R1)=0FFH；如果 Cy=0，说明(R0)>10H，则令(R1)=0AAH。

参考程序：

```
    MOV     A，R0
    CJNE    A，#10H，L1
    MOV     R1，#0
    LJMP    L3
L1：JC      L2
    MOV     R1，#0AAH
```

```
        LJMP    L3
L2: MOV     R1, #0FFH
L3: LJMP    L3
```

3) 减 1(循环)条件转移指令

减 1(循环)条件转移指令如下：

```
DJNZ    Rn, rel         ; (Rn)-1→(Rn), 若(Rn)≠0, 则(PC)+2+rel→(PC)
DJNZ    direct, rel     ; (direct)-1→(direct), 若(direct)≠0, 则(PC)+3+rel→(PC)
```

减 1 条件转移指令是把减 1 与条件转移两种功能结合在一起的指令。每执行这种指令时，先把第一操作数的内容减 1，并把减 1 后的结果仍保存在第一操作数中，然后判断结果是否为 0。若不为 0，则转移到指定的地址单元，否则顺序执行。

这组指令对于构成循环程序是十分有用的，可以指定任何一个工作寄存器作为程序循环计数器。每循环一次，这种指令被执行一次，计数器就减 1。预定的循环次数不到，计数器不会为 0，继续执行循环操作；到达预定的循环次数，计数器就被减为 0，顺序执行下一条指令，也就结束了循环。

【例 3-35】 试编写程序，将片内 RAM 以 DAT 为起始地址的 10 个单元中的数据求和，并将结果送入 SUM 单元。假设和不大于 255。

分析： 该例涉及 10 个存储单元的求和问题，最好采用循环语句实现其功能，并且可以利用 DJNZ 指令或 CJNE 指令实现循环相加的过程。因为这两个指令特点不同，因此程序的实现过程也有所区别，如例 3-32 和例 3-33，本例分别利用这两个指令编写程序。

参考程序 1：

```
        MOV     A, #00H         ; 将累加器清 0
        MOV     R0, #DAT        ; 起始地址送 R0
        MOV     R7, #0AH        ; 求和单元数送 R7
LOOP:   ADD     A, @R0          ; 求和，结果送入 A 中
        INC     R0              ; 地址加 1(循环修改)
        DJNZ    R7, LOOP        ; 若(R7)-1≠0, 程序转至 LOOP 处
        MOV     SUM, A          ; 求和的结果送入 SUM 单元保存
```

参考程序 2：

```
        MOV     A, #00H
        MOV     R0, #DAT
        MOV     R7, #00H                ; 求和单元数清零
NEXT:   CJNE    R7, #0AH, LOOP          ; 若相减结果≠0, 程序继续循环
        LJMP    ENDP
LOOP:   ADD     A, @R0                  ; 求和，结果送入 A 中
        INC     R0
        INC     R7                      ; 求和单元数加 1
        LJMP    NEXT
ENDP:   MOV     SUM, A
```

3. 子程序调用与返回指令

子程序是一种重要的程序结构，它可以在程序中反复多次使用，且能减少程序所占的存储空间。为了实现主程序对子程序的一次完整调用，主程序应该能在需要时通过调用指令自动转入子程序执行，子程序执行完后应能通过返回指令自动返回调用指令的下一条指令（该指令地址被称为断点地址）执行。因此，调用指令是在主程序需要调用子程序时使用的，返回指令则需放在子程序的末尾。**调用和返回指令必须是成对使用的。**调用指令具有把断点地址保护到堆栈以及把子程序入口地址自动送入 PC 的功能；返回指令则具有能把堆栈中的断点地址自动恢复到 PC 的功能。

1）调用指令

调用指令如下：

```
LCALL    addr16    ;(PC)+3→(PC), (SP)+1→(SP), (PC)7~0→((SP)),
                    (SP)+1→(SP), (PC)15~8→((SP)), addr16→(PC)
ACALL    addr11    ;(PC)+2→(PC), (SP)+1→(SP), (PC)7~0→((SP)),
                    (SP)+1→(SP), (PC)15~8→((SP)), addr11→(PC)10~0
```

这两条指令分别可以实现子程序的长调用和短调用，目标地址的形成方式与 LJMP 和 AJMP 指令类似，不同之处在于：在转移前要把 PC 的当前值自动压入堆栈后，才将子程序入口地址 addr16（或 addr11）送入 PC，并且调用指令遇到 RET 后结束并返回。

LCALL 与 LJMP 一样提供 16 位转移目标地址，可调用 64KB 范围内的子程序。由于该指令为 3 字节指令，所以执行该指令时先执行(PC)+3→(PC)，以获得 PC 当前值，然后将其压入堆栈（先压入低字节，后压入高字节）作为返回地址，最后再把目的地址 addr16 送入 PC，形成子程序的入口地址。

ACALL 与 AJMP 一样提供 11 位地址，只能调用与 PC 当前值在同一 2 KB 范围内的子程序。由于该指令为 2 字节指令，所以执行该指令时先执行(PC)+2→(PC)，以获得 PC 当前值，并把该地址压入堆栈作为返回地址，然后由 PC 当前值的高 5 位与指令中提供的 11 位直接地址形成子程序的目标地址。本指令也是为能与 MCS-48 的 JMP 指令兼容而设的。

2）返回指令

返回指令如下：

```
RET     ;((SP))→(PC)15~8, (SP)-1→(SP), ((SP))→(PC)7~0, (SP)-1→(SP)
RETI    ;((SP))→(PC)15~8, (SP)-1→(SP), ((SP))→ (PC)7~0, (SP)-1→(SP)
```

子程序执行完后，程序应返回到原调用指令的下一指令处继续执行，因此，在子程序的结尾必须设置返回指令 RET。中断服务程序执行完后，程序应返回到断点的下一指令处继续执行，因此，在中断服务程序的结尾必须设置返回指令 RETI。

从功能上看，这两条指令完全相同，都是把堆栈中断点地址恢复到 PC 中，从而使单片机回到断点处执行程序。但这两条指令具有以下区别：

① 在使用方面，RET 指令必须作子程序的最后一条指令；RETI 必须作中断服务程序的最后一条指令。两者不能互换使用。

② RETI 指令除恢复断点地址外，还恢复 CPU 响应中断时硬件自动保护的现场信息，如将清除中断响应时所置 1 的优先级状态触发器，使得已申请的同级或低级中断申请可以

响应；而 RET 指令只能恢复返回地址。

③ 在确定性方面，RET 指令的返回地址对开发者来说，是确知的，而 RETI 指令的返回地址对开发者来说，是未知的。

4. 空操作指令

空操作指令如下：

```
NOP     ；(PC)+1→(PC)
```

空操作指令是一条单字节单周期指令，操作码为 00H。执行该指令除了使 PC 的内容加 1 并消耗一个机器周期的时间以外，不产生任何操作结果，也不影响任何标志位。NOP 指令常用于分割程序的不同功能模块或短时间的延时。

3.3.5 位操作类指令

MCS－51 单片机的特色之一就是具有很强的位处理功能，它借用进位标志位 Cy 作为位累加器，用片内 RAM 20H～2FH(字节地址)单元中的 128 个位作为位 RAM。另外，因为它拥有自己的位 I/O 口(P0.0～P0.7，P1.0～P1.7，P2.0～P2.7 和 P3.0～P3.7)，所以某些工业控制场合中对开关量的处理(比如继电器的吸合)就变得非常简便。

位操作类指令的操作数不是字节，而是字节中的某个位，每位只能取 0 或 1，所以位操作类指令又称为布尔变量操作类指令。MCS－51 单片机的指令系统中的位操作类指令分为位传送指令、位状态设置指令、位逻辑运算指令和位条件转移指令 4 种，从而完成以位为对象的数据传送、运算、控制转移等操作。

1. 位传送指令

位传送指令如下：

```
MOV     C，bit     ；(bit)→Cy
MOV     bit，C     ；Cy→(bit)
```

其功能是实现位累加器 Cy 和其他位地址之间的数据传送。

注意：位传送指令必须以 Cy 作为其中的一个操作数，两个位地址间不能直接进行传送。

在程序中，Cy 记作 C。

【例 3－36】 试编写程序将位地址 10H 的内容与位地址 20H 的内容交换。

分析： 因为两个位地址间不能直接进行传送，且位传送指令中必须以 Cy 作为其中的一个操作数，所以对于单向传送的情况，直接通过 Cy 作为中转即可；但对于双向传送的情况，需要通过 Cy 和其他位存储单元配合作为中转。比如本例的情况，要先将 10H 的内容暂存至 Cy，再将 Cy 的内容送入 PSW 中的 F0 位内暂时保存，然后通过 Cy 位将 20H 位中的内容送入 10H 位中，再将 F0 位中的内容(原 10H 位中的内容)送回 Cy 位中，并通过 Cy 将原 10H 位中的内容送到 20H 位中。该情况与进行片外 RAM 存储单元的内容交换类似。

参考程序：

```
MOV     C，10H
MOV     F0，C
MOV     C，20H
```

```
MOV    10H, C
MOV    C, F0
MOV    20H, C
```

2. 位状态设置指令

1）位清0指令

位清0指令如下：

```
CLR    C      ; 0→Cy
CLR    bit    ; 0→(bit)
```

其功能是将Cy或指定位地址的内容清0。

2）位置1指令

位置1指令如下：

```
SETB   C      ; 1→Cy
SETB   bit    ; 1→(bit)
```

其功能是将Cy或指定位地址的内容置1。

3. 位逻辑运算指令

1）位逻辑与指令

位逻辑与指令如下：

```
ANL    C, bit     ; Cy∧(bit)→Cy
ANL    C, /bit    ; Cy∧(/bit)→Cy
```

其功能是将源操作数指定的位地址内容和位累加器Cy的内容进行逻辑与运算，运算结果存于Cy中。其中“/”表示对位单元内容取反后再进行逻辑运算。

2）位逻辑或指令

位逻辑或指令如下：

```
ORL    C, bit     ; Cy∨(bit)→Cy
ORL    C, /bit    ; Cy∨(/bit)→Cy
```

其功能是将源操作数指定的位地址内容和位累加器Cy的内容进行逻辑或运算，运算结果存于Cy中。

3）位取反指令

位取反指令如下：

```
CPL    C      ; /Cy→Cy
CPL    bit    ; (/bit)→(bit)
```

其功能是将位累加器Cy或指定的位地址内容取反。

【例3-37】 试编程实现如图3-14所示的逻辑电路的功能。

参考程序：

```
MOV    C, X
ANL    C, Y     ; 将X和Y“相与”后送入Cy
MOV    F0, C    ; 送F0保存
```

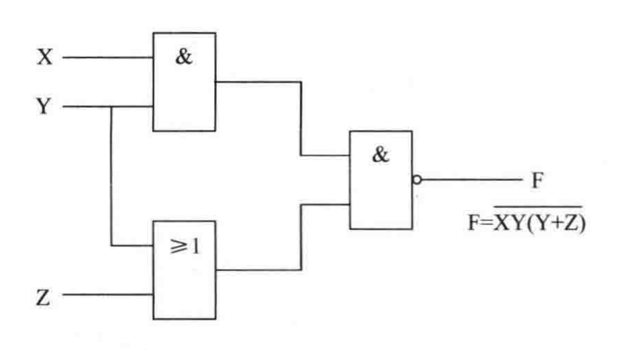


图 3-14　例 3-37 逻辑电路图

```
MOV     C, Y
ORL     C, Z        ; 将 Y 和 Z"相或"后送入 Cy
ANL     C, F0       ; 再将 Y+Z 和 XY"相与"
CPL     C           ; 最后取"非"
```

参考程序中的 X、Y、Z 实际上应该用伪指令进行定义(见 4.1.2 小节)，在此为了突出位运算而将其省略了。下面的例 3-38 情况相同。

【例 3-38】 试编写程序，将位 M 和位 N 的内容相异或，结果存入 F0 位中。

分析： 位运算中没有"异或"运算，需用 $M\oplus N=M\overline{N}+\overline{M}N$ 来进行"异或"运算。

参考程序：

```
MOV     C, M        ; 将位 M 中的内容送入 Cy
ANL     C, /N       ; 将 M 和 N̄ "相与"后的内容送入 Cy
MOV     F0, C       ; 再暂存于 F0 位中
MOV     C, N        ; 将位 N 中的内容送入 Cy
ANL     C, /M       ; 将 N 和 M̄ "相与"后的内容送入 Cy
ORL     C, F0       ; 将 MN̄+M̄N 送入 C
MOV     F0, C       ; 最后结果存于 F0 位中
```

4. 位条件转移指令

位条件转移指令有两组：第 1 组根据进位标志位 Cy 的状态进行判断转移；第 2 组以位地址 bit 的内容作为转移的条件。

1) 判 Cy 状态转移指令

判 Cy 状态转移指令如下：

```
JC      rel     ; 若 Cy=1, 则(PC)+2+rel→(PC)
JNC     rel     ; 若 Cy=0, 则(PC)+2+rel→(PC)
```

第一条指令的功能是：如果 Cy=1，则程序发生转移，否则程序顺序执行。

第二条指令的功能是：如果 Cy=0，则程序发生转移，否则程序顺序执行。

这两条指令都是相对转移指令，都是以 Cy 中的值来决定程序是否需要转移。因此这组指令常常和比较条件转移指令 CJNE 连用，以便根据 CJNE 指令执行过程中形成的 Cy

进一步决定程序的流向或形成三分支模式。

2）判位内容转移指令

判位内容转移指令如下：

```
JB      bit, rel     ；若(bit)=1，则(PC)+3+rel→(PC)
JNB     bit, rel     ；若(bit)=0，则(PC)+3+rel→(PC)
JBC     bit, rel     ；若(bit)=1，则(PC)+3+rel→(PC)，且 0→(bit)
```

第一条指令的功能是：如果(bit)=1，则程序发生转移，否则程序顺序执行。

第二条指令的功能是：如果(bit)=0，则程序发生转移，否则程序顺序执行。

第三条指令和第一条指令相同，即(bit)=1，则程序发生转移，但在转移之前，把该位的值清 0，否则程序顺序执行。

【例 3-39】 P3.2 和 P3.3 上各接有一只按键，如图 3-15 所示，要求它们分别按下时(P3.2=0 或 P3.3=0)，分别使 P1 口输出 0 或 FFH，试编程实现该功能。

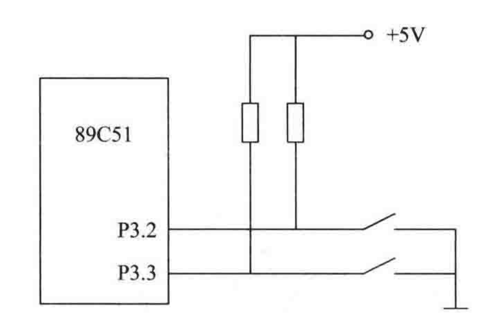


图 3-15　例 3-39 电路图

分析： 因为 P3 是准双向口，所以在读取按键状态之前，要先通过程序使 P3 口输出高电平，然后再依次读取 P3.2 和 P3.3 的状态并进行判断：如果 P3.2=1，P3.3=1，则说明两个按键没有被按下，程序处于等待状态；如果 P3.2=0，则 P1 口 8 位全输出“0”；如果 P3.3=0，则 P1 口 8 位全输出“1”。

参考程序：

```
        MOV     P3, #0FFH       ；P3 口作输入口之前，先将 P3 口置“1”
L1:     JNB     P3.2, L2        ；判断 P3.2 键是否按下
        JNB     P3.3, L3        ；判断 P3.3 键是否按下
        LJMP    L1
L2:     MOV     P1, #00H        ；如果 P3.2 键按下，则 P1 口 8 位输出“0”
        LJMP    L1
L3:     MOV     P1, #0FFH       ；如果 P3.3 键按下，则 P1 口 8 位输出“1”
        LJMP    L1              ；返回，继续判断 P3.2，P3.3
```

思考与练习

1. MCS-51 指令系统按功能可以分为哪几类？每类指令的功能分别是什么？

2. MCS－51 单片机有哪几种寻址方式？并指出下列每条指令的寻址方式。

(1) MOV　C，20H

(2) MOV　A，60H

(3) JZ　LOOP

(4) MOV　A，@R1

(5) MOVC　A，@A＋DPTR

(6) MOV　A，R1

(7) MOV　A，＃60H

3. 判断下列指令的对错，错的请说明原因并改正。

(1) MOV　R1，R0

(2) MOV　＃20H，60H

(3) MOVX　A，@R0

(4) MOV　A，@R2

(5) MOVX　@DPTR，20H

(6) MOV　@R0，50H

(7) PUSH　A

(8) ADD　20H，A

(9) MOV　A，@DPTR

(10) MUL　A，B

4. 若(40H)＝30H，请分析执行以下程序段后，累加器 A、寄存器 R1 及片内 RAM 的 40H、41H 单元中的内容。

```
MOV    A，40H
MOV    R1，A
MOV    A，＃00H
MOV    @R1，A
MOV    A，＃0FFH
MOV    41H，A
```

5. 若(A)＝68H，(R0)＝30H，(R1)＝50H，(R2)＝6AH，(30H)＝2FH，(50H)＝F0H，试分析下列各指令独立执行后，有关寄存器和存储单元的内容的变化。

(1) MOV　A，@R0

(2) XCH　A，@R1

(3) ADD　A，R2

(4) INC　A

(5) SWAP　A

(6) MOV　R1，A

(7) XRL　30H，＃0FH

6. 若 Cy＝1，(P0)＝10100011B，(P2)＝01101100B，试分析下列程序段执行过程中，Cy、P0 口及 P2 口的内容变化情况。

```
    MOV     P0.3, C
    MOV     P0.4, C
    MOV     C, P0.6
    MOV     P2.6, C
    MOV     C, P0.0
    MOV     P2.4, C
```

7. 已知(31H)＝36H，(32H)＝39H，试分析下列程序段执行过程中，有关寄存器和存储单元的变化情况。

```
START: MOV     A, 31H
       ANL     A, #0FH
       SWAP    A
       MOV     30H, A
       MOV     A, 32H
       ANL     A, #0FH
       ORL     30H, A
       LJMP    $
```

8. 已知(A)＝10011100B，(R0)＝30H，试分析下列程序段执行过程中，有关寄存器和存储单元的变化情况。

```
    MOV     B, #100
    DIV     AB
    MOV     @R0, A
    DEC     R0
    MOV     A, #10
    XCH     A, B
    DIV     AB
    SWAP    A
    ADD     A, B
    MOV     @R0, A
    INC     R0
```

9. 试用位操作指令或逻辑运算指令实现下列逻辑操作。

(1) 令片内 RAM 40H 单元中数据的低两位变为“0”，其余位不变；

(2) 令片内 RAM 40H 单元中数据的低两位变为“1”，其余位不变；

(3) 令片内 RAM 40H 单元中数据的低两位变反，其余位不变；

(4) 令 ACC.3、ACC.4 变为“0”，其余位不变。

第 4 章　单片机汇编语言程序设计

本章重点讲述 MCS－51 单片机汇编程序编程方法，从编程思路、编程技巧等方面做了细致讲解，并给出了一些常用子程序。本章在例题中只使用了最常用的 88 条指令(在附录 B 中标注“＊”的指令)，从实践来看，这 88 条指令是学习的重点，也可以覆盖所有的开发工作。

4.1　汇编语言程序的设计基础

虽然单片机的每一条指令能使计算机完成一种特定的操作，但要完成某一特定的任务还需要将这些指令按任务要求有序组合成一段完整的程序。**程序实际上是一系列计算机指令的有序集合**。我们把利用计算机指令系统来合理地编写出解决某个问题的程序的过程，称为程序设计。

程序设计是单片机应用系统设计的重要组成部分，单片机的全部动作都是在程序的控制下进行的。随着芯片技术的发展，很多标准的或功能型的硬件电路都集成到了芯片中。所以，程序设计在单片机应用系统开发中占的比重越来越大，这一点在有驻留系统程序的高级嵌入式系统中更加明显。

4.1.1　汇编语言的语句格式

汇编语言的语句有两种基本类型：指令语句和伪指令语句。

1. 指令语句

指令语句是引发单片机操作的命令。每一条指令语句在汇编时都产生一个指令代码(也称机器代码)，执行该指令代码对应着单片机的一种操作。

2. 伪指令语句

伪指令语句是控制汇编(翻译)过程的一些控制命令。在汇编时没有机器代码与伪指令语句对应。

4.1.2　伪指令

汇编时为了便于汇编器(某些情况下，也可以称为编译器)的操作，汇编程序提供了一些本身的操作命令，比如汇编器汇编时需要知道汇编语言源程序中哪些是数据、数据的状态、程序的起始和终止等。这些汇编器本身的操作指令可以出现在汇编语言源程序中，但它不控制单片机操作，而是控制汇编器的指令。这些指令称为伪指令。

伪指令是程序员发给汇编器的命令，也称为汇编命令或汇编程序控制指令。

MCS－51 汇编语言程序中常用的伪指令有以下几个。

(1) ORG(ORiGin)汇编起始地址指令：

```
ORG    ＜地址＞
```

其中，"＜地址＞"是 4 位十六进制数，这个十六进制数前不用加"#"，这点与汇编指令中的立即数前加"#"不同，需要注意。

ORG 指令的功能是通知汇编器在把程序"翻译"成机器码的过程中，将 ORG 后面这段程序的起始地址设定在 ORG 规定的地址上。MCS-51 系列单片机执行的机器码是放在程序存储器中的，执行是从首地址开始的。通常，我们在编写一段真正可以在实际系统上运行的程序时，第一条程序应该是

```
ORG    0000H
```

注意：ORG 所定义不同程序段的地址不能产生代码区重叠。

(2) END(END of assembly) 汇编终止指令：

```
END
```

END 指令的功能是通知汇编器在把程序"翻译"成机器码的过程中，遇到 END 就停止"翻译"。这条语句是给汇编器的指令，而不是给单片机的指令，不表示程序运行到这里就结束。

END 指令是单指令，一般没有标号和表达式。

(3) EQU(EQUate)赋值指令：

```
＜字符名称＞    EQU    ＜赋值项＞
```

EQU 指令用于将一个数值或寄存器名赋给一个指定符号名。经过 EQU 指令赋值的符号可在程序的其他地方使用，以代替其赋值。例如：

```
MAX    EQU    200
```

表示在程序的其他地方出现 MAX 时用 200 代替。

(4) DB(Define Byte) 定义数据字节指令：

```
[＜标号：＞]    DB    ＜8 位二进制数表＞
```

DB 指令的功能是给定表达式的值以字节形式初始化代码空间。通常是在程序存储器的某个位置预置一个字节数(8 位二进制数)，可以和 ORG 配合使用；"＜8 位二进制数表＞"前不加"#"，一般用 2 位十六进制数表示。例如：

```
ORG    1000H
DB     24H
```

即在程序存储器地址 1000H 的位置，放入一个数值为"24H"的数。

可以多个字节同时预置，用逗号分隔。例如：

```
TAB:    DB    30H, 31H, 32H, …
```

也可以预置一个字符串，用双引号括起来，字符自动以 ASCII 码形式存放。例如：

```
TAB:    DB    "CPU"
```

相当于“TAB:　DB　43H，50H，55H”，这三个数为“CPU”三个字母的 ASCII 码。

(5) DW(Define Word) 定义数据字指令：

```
[<标号:>]    DW    <16 位二进制数表>
```

DW 指令的功能是给定表达式的值以字形式(双字节)初始化代码空间。通常是在程序存储器的某个位置预置两个字节数(16 位二进制数)，可以和 ORG 配合使用；“<16 位二进制数表>”前不加“#”，一般用 4 位十六进制数表示。DW 的使用方法和 DB 类似。例如：

```
ORG    0500H
DW     1234H
```

(6) DATA 数据地址赋值指令：

```
<字符名称>    DATA    <表达式>
```

DATA 指令用于将一个内部 RAM 的地址赋给指定的符号名。数值表达式的值应在 00H～FFH(0～255)之间。例如：

```
BUF1    DATA    40H
BUF2    DATA    80H
```

DATA 伪指令的使用与 EQU 相似，不同之处是：

- EQU 定义的标识符必须先定义后使用，DATA 定义的符号名可以先使用后定义；
- EQU 可以把汇编符号赋予标识符，而 DATA 则不可以。

(7) BIT 位定义指令：

```
<字符名称>    BIT    <位地址>
```

BIT 指令用于将一个位地址赋给指定的符号名。经 BIT 指令定义过的位符号名不能更改。例如：

```
X_ON     BIT    60H
X_OFF    BIT    24H.2
```

(8) DS(Define Stonage) 定义存储区指令：

```
[<标号:>]    DS    <数表>
```

DS 指令的功能是以字节为单位在内部和外部存储器保留存储空间，标号值将是保留区的第一个字节地址。例如：

```
ORG            0200H
COUNTER        DS        10
```

即 COUNTER 的首地址是 0200H，长度是 10。

4.1.3 汇编语言程序的结构

完成控制任务的汇编语言源程序基本上由主程序、子程序、中断服务程序等组成。单片机程序存储器的某些单元被保留作为特定的程序入口地址，0000H 是单片机的启动地

址。由于系统复位后的 PC(程序计数器)的内容是 0000H，所以系统从 0000H 开始取指令，执行程序。如果程序需要用到中断功能，则程序必须在 0000H 开始的第一条指令放入一个无条件转移指令，将主程序引到别处。程序段的起始定位可由伪指令 ORG 来规定，主程序的开始地址最好设在 0030H 以后。

0003H、000BH、0013H、001BH、0023H 分别是五个中断服务程序的入口地址。当中断产生时，PC 会自动指向这里，即程序可以从这五个地址的某一个开始运行。由于每一入口的地址空间较小，一般只有几个字节，通常在入口地址存放一条无条件转移指令，将中断服务程序引到别处。

单片机标准汇编程序结构如图 4－1 所示。

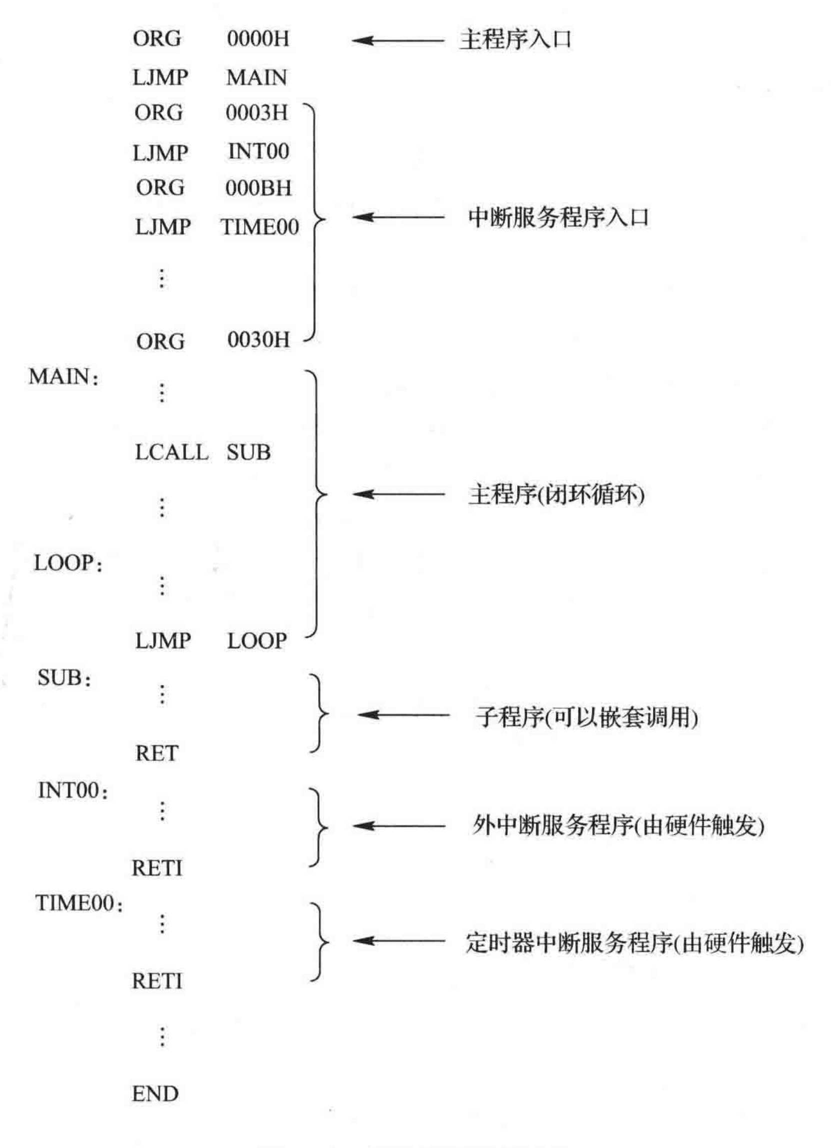


图 4－1　标准汇编程序结构

注意事项：

① 必须有主程序，程序入口地址是 0000H。如果有中断服务程序，主程序必须是一个跳转指令，将主程序引出中断向量区，一般可以从 0030H 开始。

② **必须用伪指令来规定程序的开始地址**。除了主程序，其他一些程序段也可以用伪指令来规定开始地址，可能会形成一些不连续的代码区，尤其要注意后面的代码区不能覆盖前面的代码区。

③ 用户可以根据功能要求设计子程序。

④ 如果使用中断、定时器、串口等功能，则单片机相关寄存器必须要初始化。

⑤ **主程序必须是闭环结构**，即是一个闭环循环，采用 LOOP：… LJMP　LOOP(类似 C 语言例程中的 while(1){…})表示单片机的执行代码有一部分是循环执行。实际应用中，单片机程序先进行一次初始化操作，然后使用循环执行其执行代码部分。后面的例程由于部分主程序没有实际功能，本书的例题中一般用一条“LJMP　$”来代替循环执行代码。

⑥ 注释部分可以用“；注释内容”或“// 注释内容”来分割。

4.1.4　汇编语言程序的编辑与汇编

汇编语言源程序的编写要在一个编辑工具上进行，例如，“写字板”、“Keil 集成开发环境”(见第 9 章)等。编写好的程序，单片机是不“认识”的，单片机能够认识的就是由“0”和“1”组成的机器码。汇编指令编写的汇编语言源程序，需转换成二进制代码表示的机器语言程序，单片机才能识别和执行，通常把这一转换(翻译)工作称为“汇编”。

完成“翻译”工作的专用程序称为汇编程序软件(即汇编器)，经汇编程序“汇编”得到的以 0、1 代码形式表示的机器语言程序称为目标程序，这一过程通常是在微机上进行的。

汇编程序从编写到生成可执行机器码的过程如图 4－2 所示。其中，HEX 文件(十六进制文件)是由 Intel 公司定义的一种格式，包括地址、数据和校验码，用 ASCII 码存储，可以显示和打印，通常用于仿真器的集成调试环境界面显示(详见第 9 章)。还有一种是 BIN(二进制)格式的文件，它完全是由编译器生成的二进制文件，是程序的机器码。两种格式都支持写入单片机或仿真器调试的目标程序。

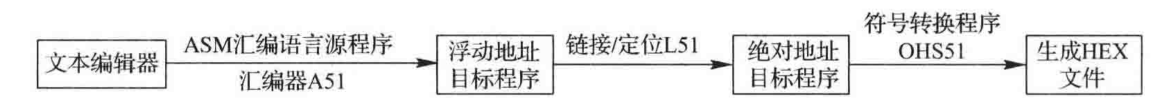


图 4－2　汇编程序从编写到生成可执行机器码的过程

目前，很多公司已将编辑器、汇编器、编译器、链接/定位器、符号转换程序等做成集成软件包，如 Keil μVision。

4.1.5　汇编语言程序的设计方法

用汇编语言进行程序设计的过程和用高级语言进行程序设计相类似。在进行程序设计时，要按照实际问题的要求和单片机的特点，决定所采用的设计方法、逻辑关系、计算公式和步骤，也就是通常所说的算法。合适的算法常常可以起到事半功倍的效果，然后根据单片机的指令系统来编制程序。

1. 任务分析(硬件、软件系统分析)

根据单片机系统需要完成的控制任务，在确定好的单片机的硬件体系、外围电路条件下，结合软件的功能分配，明确软件应该完成的任务。

2. 确定算法和工作步骤

对一些复杂的任务，需要提炼数学算法，提高编程质量，当然这些还要和硬件完全配合。也可以说，不同的硬件也许有不同的算法。但对于一些简单的功能(多数情况)，则不需要复杂算法，只要把逻辑关系弄明白即可。编程前，还需考虑整个程序的功能模块划分。

3. 制定程序流程图

流程图也称为程序框图，就是用各种符号、图形、箭头把程序的流向及过程用图形表示出来，使程序清晰、结构合理、便于调试。绘制流程图是编写单片机程序前最重要的工作，通常程序就是根据流程图的指向采用适当的指令来编写的。流程图一般可以用绘图工具绘制，常用工具有 Visio、AutoCAD 等，流程图也可以手工绘制，但不便于保存和修改。图 4－3 是流程图的常用格式。

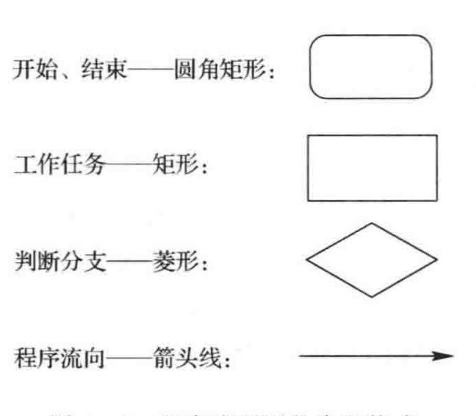


图 4－3　程序流程图的常用格式

4. 分配内存，确定程序与数据区存放地址

分配内存工作要根据程序区、数据区、堆栈区等预计所占空间大小，对片内外存储区进行合理分配并确定每个区域的首地址，便于编程使用。

5. 编写源程序

根据任务要求和程序流程图编写程序代码。尽可能按照节省数据存放单元、缩短程序长度和减少运算时间三个原则来编制程序。

6. 调试、修改，最终确定程序

通过仿真器或用直接方法进行调试，即发现错误和修改错误的过程。程序错误分为两类：一类是语法错误，比较容易发现，一般编译时就可以发现；另一类是逻辑错误，如指令使用错误、指令缺失、算法错误、顺序安排错误等，很难发现，需要在调试过程中根据中间结果或其他现象来判断，有时甚至需要利用仿真器的单步执行方法去调试(详见 9.2.5 节)。

4.2　汇编语言程序的基本结构形式

汇编语言程序设计一般采用模块化编程，基本程序结构有三种：顺序结构、分支结构和循环结构。这三种结构组合起来，可以解决任何复杂问题。

4.2.1 顺序程序

顺序结构是最简单、最基本的程序结构，其特点是按指令的排列顺序一条一条地执行，直到全部指令执行完毕为止。不管多么复杂的程序，总是由若干顺序程序段所组成的，如果某一个需要解决的问题可以分解成若干个简单的操作步骤，并且可以由这些操作按一定的顺序构成一种解决问题的算法，则可用简单的顺序结构来进行程序设计。

【例 4-1】 三字节无符号数相加，其中被加数在内部 RAM 的 50H、51H 和 52H 单元中(低位在后)，加数在内部 RAM 的 53H、54H 和 55H 单元中(低位在后)；要求把相加之和存放在 50H、51H 和 52H 单元中(低位在后)，进位存放在位寻址区的 00H 位中。

分析 1：单片机累加器 A 是 8 位的，本例是 3 字节 24 位的加法，这就需要分成三次从低到高使用加法指令进行加法运算。需要注意的是：最低 8 位相加时，无需考虑进位，可以采用 ADD 指令，其他部分因为需要考虑进位位，所以采用 ADDC 指令。

这里采用被加数放到累加器 A 中，再直接和内存单元中的加数之间求和，比较直接，但程序将来的扩展性不好。

参考程序 1：

```
ORG     0000H
MOV     A, 52H
ADD     A, 55H      ；低字节相加
MOV     52H, A      ；存低字节相加结果
MOV     A, 51H
ADDC    A, 54H      ；中字节带进位相加
MOV     51H, A      ；存中字节相加结果
MOV     A, 50H
ADDC    A, 53H      ；高字节带进位相加
MOV     50H, A      ；存高字节相加结果
MOV     00H, C      ；进位送 00H 位保存
END
```

分析 2：可以考虑采用寄存器间接寻址来获取操作数，这样程序将来便于转化成循环方式(见例 4-7)，需要统一使用 ADDC 指令，因此在进行最低 8 位相加之前，先清除进位位(进位位清“0”)。

参考程序 2：

```
ORG     0000H
CLR     C               ；后面直接用 ADDC 指令，这里需要清进位位
MOV     R0, #52H        ；被加数的低字节地址
MOV     R1, #55H        ；加数的低字节地址
MOV     A, @R0
ADDC    A, @R1          ；低字节相加
MOV     @R0, A          ；存低字节相加结果
DEC     R0
DEC     R1
```

```
MOV     A, @R0
ADDC    A, @R1        ；中间字节带进位相加
MOV     @R0, A        ；存中间字节相加结果
DEC     R0
DEC     R1
MOV     A, @R0
ADDC    A, @R1        ；高字节带进位相加
MOV     @R0, A        ；存高字节相加结果
MOV     00H, C        ；进位送 00H 位保存
END
```

4.2.2　分支程序

在实际程序设计过程中，会有很多复杂状态和条件，需要根据这些条件进行不同的选择，这时就必须对某一变量的状态进行判断，根据判断结果选择不同的程序流向，这就是分支程序。**分支程序有单分支程序、双分支程序、多分支(散转)程序**。

在 MCS－51 单片机指令系统中，有 JZ(JNZ)、CJNE、JC(JNC)及 JB(JNB)等丰富的控制转移指令，它们是分支结构程序设计的基础，可以完成各种各样的条件判断，控制转移方向。

1. 单分支程序

有一个控制转移方向的程序是单分支程序。

【例 4－2】 两个无符号数比较(单分支)。内部 RAM 的 30H 单元和 40H 单元各存放了一个 8 位无符号数，比较这两个数的大小。若(30H)≥(40H)，则将地址为 20 的内存单元置 0；否则，将地址为 20H 的内存单元置 1。

分析： 程序比较很简单，流程图如图 4－4 所示。

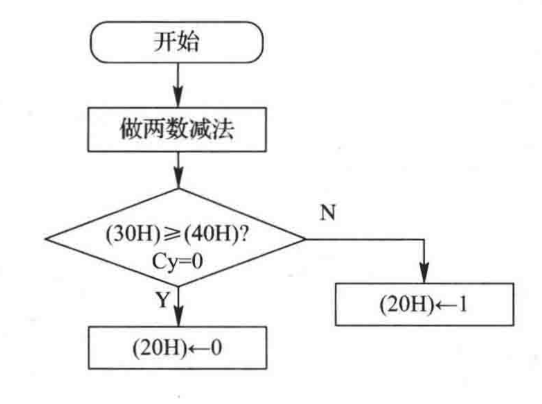


图 4－4　例 4－2 的程序流程图

参考程序：

```
ORG     0000H
MOV     A, 30H        ；被减数放入累加器中
CLR     C
```

```
        SUBB    A, 40H          ; 两数相减
        JC      L1              ; 有进位(借位)，则前小后大，转到 L1
        MOV     20H, #00H
        LJMP    L2              ; 不能执行下一条语句，需转走
L1:     MOV     20H, #01H
L2:     LJMP    $               ; 与"L2: LJMP  L2"语句相同
        END
```

2. 多分支程序

有两个或两个以上控制转移方向的程序是多分支程序。

【例 4-3】 变量 X 存放在 UNIT1 单元内，函数值 Y 存放在 UNIT2 单元中，试按下式的要求给 Y 赋值。

$$Y=\begin{cases}1 & X>0\\ 0 & X=0\\ -1 & X<0\end{cases}$$

分析 1： 该例涉及正数、负数和 0 的判断，符号位在最高位 ACC.7。可以按照如下思路进行编程：

将存放在 UNIT1 单元的 X 传送到累加器 A，再利用 JZ 或者 JNZ 指令判断该数是否为 0。如果(A)=0，则将 0 送入 Y 中(即 UNIT2 单元)；如果(A)≠0，则需要提取符号位进一步判断 A 的正负性。根据"逻辑与"操作的性质，只要将 A 与 1000 0000B 进行逻辑与操作，即可保留最高位(符号位)，屏蔽低 7 位，然后再根据 A 的值进行判断：执行上述逻辑与操作后，(A)≠0，则最高位为 1，代表该数为负数，此时将−1 以补码形式送入 Y 中，−1 的补码是 FFH；否则(A)=0，即最高位为 0，代表该数为正数，此时将+1 送入 Y 中。

图 4-5(a)实际是三分支而归一的流程图，至少要用两个转移指令。

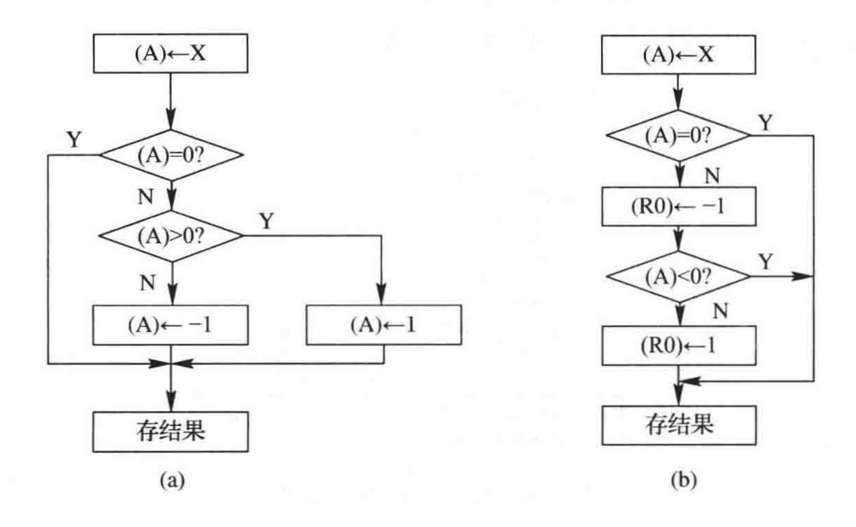


图 4-5　例 4-3 的分支流程图

参考程序 1：

```
        ORG     0000H
UNIT1   EQU     30H             ; 需要伪指令对 UNIT1 指定一个内存单元
```

```
        UNIT2   EQU  31H        ；同上
        MOV     A，UNIT1        ；(A) ← X
        JZ      DONE            ；若X=0，则转DONE
        JNB     ACC.7，POSI     ；若X>0，则转POSI
        MOV     A，#0FFH        ；若X<0，则Y=-1，用补码，-1的补码是FFH
        LJMP    DONE            ；需要转走，避免A重新赋值
POSI：  MOV     A，#01H         ；若X>0，则Y = 1
DONE：  MOV     UNIT2，A        ；存函数值
        LJMP    $
        END
```

初学者很容易犯的一个错误是：漏掉了其中的“LJMP　DONE”语句，因为流程图中没有明显的转移痕迹。

分析2：这个程序也可以按图4-5(b)所示的流程图来编写，其特征是先赋值，再比较判断，然后修改赋值并结束。

参考程序2：

```
        ORG     0000H
        MOV     A，UNIT1       ；(A) ← X
        JZ      DONE           ；若X=0，则转DONE
        MOV     R0，#0FFH      ；先设X<0，(R0)=FFH(-1的补码)
        JB      ACC.7，NEG     ；若X<0，则转NEG
        MOV     R0，#01H       ；若X>0，则(R0)=1
NEG：   MOV     A，R0          ；若X<0，则Y=-1，X>0，Y=1
DONE：  MOV     UNIT2，A       ；存函数值
        LJMP    $
UNIT1   DATA    30H
UNIT2   DATA    31H
        END
```

在参考程序1中，使用伪指令EQU定义UNIT1和UNIT2，在参考程序2中使用伪指令DATA定义UNIT1和UNIT2，最后执行结果相同，说明在本例中两种伪指令都可以使用。

3. 散转程序

散转程序是一种并行分支程序，它可根据运算结果或输入数据将程序转入不同的分支。MCS-51指令系统中有一条散转指令“JMP @A+DPTR”，用它可以很容易地实现散转功能。该指令把累加器的8位无符号数与16位数据指针的内容相加，并把相加的结果装入程序计数器PC，控制程序转向目标地址去执行。此指令的特点在于：转移的目标地址不是在编程或汇编时预先确定的，而是在程序运行过程中动态地确定的，目标地址是以数据指针DPTR的内容为起始的256个字节范围内的指定地址，即由DPTR的内容决定

分支转移程序的首地址，由累加器 A 的内容来动态选择其中的某一个分支转移程序，如图 4-6 所示。**散转程序适合在键盘控制程序中使用。**

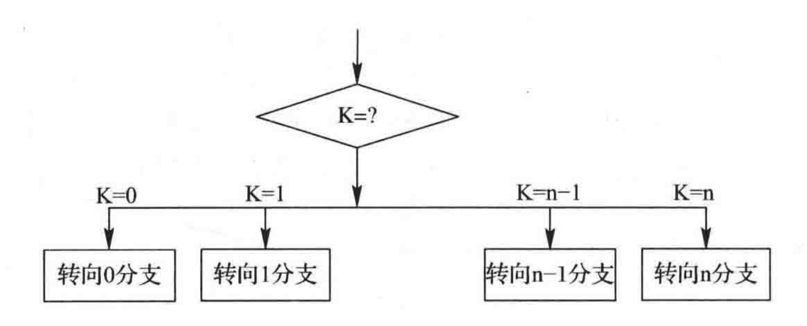


图 4-6　散转程序转移

实现程序散转常用指令表法，即在程序存储器中建立一个转移指令表，将表的首地址送 DPTR，将特定单元的内容送 A，再通过散转指令就可以根据 A 值转向转移指令表中的某条 LJMP 指令，然后再执行"LJMP　PRGi"，把程序转移到指定的分支程序入口 PRGi。这种方法实际上是通过 2 次转移(1 次执行散转指令，1 次执行无条件转移指令)实现的。

【例 4-4】 已知累加器 A 中存放着控制程序转向的编号 0～n(n＜10)，ROM 中存有起始地址为 TABLE 的三字节长转移指令表，试编程使单片机能按照累加器 A 中的编号转去执行相应的命令程序，即当(A)＝0 时，执行 PR0 分支程序；当(A)＝1 时，执行 PR1 分支程序；当(A)＝2 时，执行 PR2 分支程序；当(A)＝n 时，执行 PRn 分支程序。

分析：考虑用散转指令"JMP　@A＋DPTR"来实现，定义散转表首地址是 TAB，需注意"LJMP　<标号>"是三个字节，(A)＝0 对应分支程序标号 PR0，地址是 TAB＋0；(A)＝1 对应分支程序标号 PR1，地址是 TAB＋3；(A)＝n 对应分支程序标号 PRn，地址是 TAB＋3×n，所以用 A 求地址偏移量时需将 A 中的数值"乘以"3。有多种方法能够获取偏移量 A×3 的功能。程序流程图如图 4-7 所示。

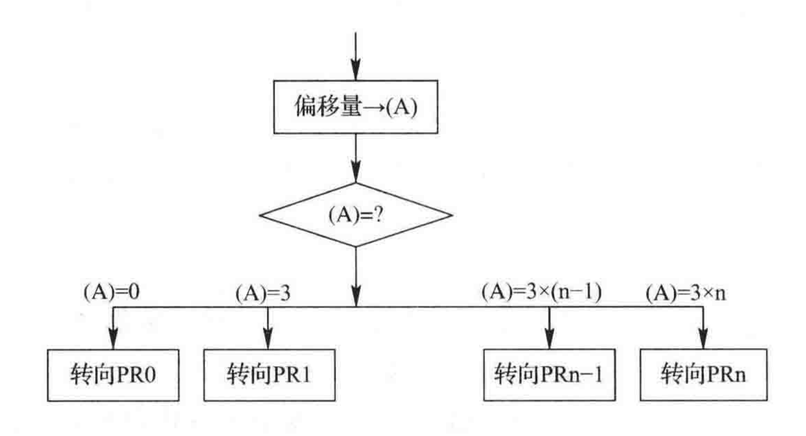


图 4-7　例 4-4 的程序流程图

参考程序 1(采用"直接相乘"得到 3×A)：

```
        CLR     C
        MOV     B，#03H
        MUL     AB                  ；得到 3×A
        MOV     DPTR，#TABLE        ；将 TABLE 地址送入 DPTR 寄存器中
        JMP     @A+DPTR             ；程序转到地址为 A+DPTR 的地方去执行
TABLE:  LJMP    PR0                 ；当(A)=0 时，执行 PR0 分支程序
        LJMP    PR1                 ；当(A)=1 时，执行 PR1 分支程序
        LJMP    PR2                 ；当(A)=2 时，执行 PR2 分支程序
        ⋮
        LJMP    PRn                 ；当(A)=n 时，执行 PRn 分支程序
```

参考程序 2(采用“三个 A 相加，即 A+A+A”得到 3×A)：

```
        CLR     C
        MOV     B，A
        ADD     A，B                ；得到 2×A
        ADD     A，B                ；得到 3×A
        MOV     DPTR，#TABLE        ；将 TABLE 地址送入 DPTR 寄存器中
        JMP     @A+DPTR             ；程序转到地址为 A+DPTR 的地方去执行
TABLE:  LJMP    PR0                 ；当(A)=0 时，执行 PR0 分支程序
        ⋮
```

参考程序 3(采用“左移一次，再加一次”得到 3×A)：

```
        CLR     C
        MOV     B，A
        RLC     A                   ；得到 2×A
        ADD     A，B                ；得到 3×A
        MOV     DPTR，#TABLE        ；将 TABLE 地址送入 DPTR 寄存器中
        JMP     @A+DPTR             ；程序转到地址为 A+DPTR 的地方去执行
TABLE:  LJM     PPR0                ；当(A)=0 时，执行 PR0 分支程序
        ⋮
```

参考程序 4(不限定 n 值范围)：

```
        MOV     DPTR，#TABLE        ；将 TABLE 地址送入 DPTR 寄存器中
        MOV     B，A
        ADD     A，B
        ADD     A，B                ；A 内容乘以 3
        JNC     LP1                 ；无进位转移，有进位位则 DPH 加“1”
        INC     DPH                 ；加进位位
LPl:    JMP     @A+DPTR             ；跳至散转表中相应位置
TABLE:  LJMP    PR0
        ⋮
```

若不限定 n 值范围，3×n 有可能会大于 256，超过了累加器 A 的存储范围，则需把进

位位考虑进来，加到 DPH 上。

4.2.3 循环程序

在程序设计过程中，经常会遇到需要重复执行某一段程序的情况，这时使用循环程序结构，可以节省程序存储空间，提高程序的质量。从本质上看，循环程序结构只是分支程序中的一个特殊形式而已。

循环程序一般由 4 部分组成：

(1) 设置循环初值，即确立循环开始时的状态。

(2) 循环体(工作部分)，要求重复执行的部分。

(3) 循环修改，循环程序必须在一定条件下结束，否则就要变成死循环。

(4) 循环控制部分，根据循环结束条件，判断是否结束循环。

循环控制的一般方法有：

① 循环次数已知，利用循环次数控制。

② 循环次数未知，利用关键字控制。

以上 4 个部分可以有两种组织方式，一种是先循环后判断，另一种是先判断后循环，如图 4－8 所示。

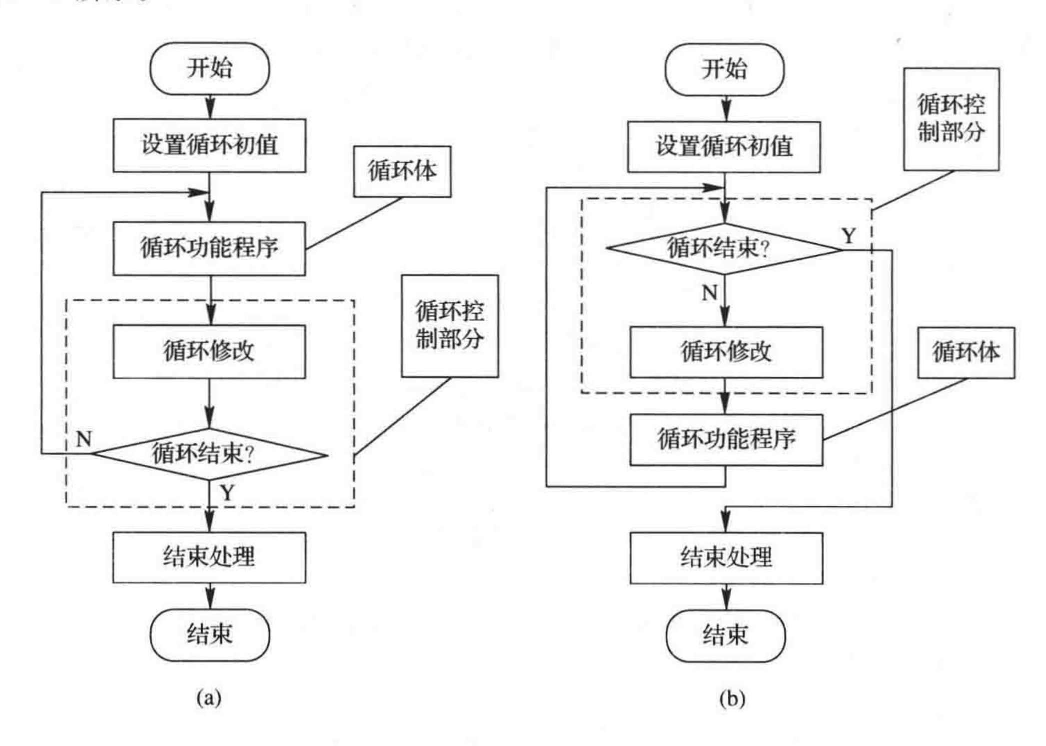


图 4－8 循环程序流程图

循环程序体中不再包含其他循环程序，即为单循环程序。如果在循环体中还有循环，则为循环嵌套，或多重循环。在多重循环中，只允许外重循环嵌套内重循环，不允许循环相互交叉，也不允许从循环程序外部跳到循环程序内部。

【例 4－5】 从 BLOCK 单元开始存放一组无符号数，一般称为一个数据块。数据块长度放在 LEN 单元，编写一个求和程序，将和存入 SUM 单元。假设和不超过 8 位二进

制数。

分析：在设置初值时，将数据块长度置入一个工作寄存器，将数据块首地址送入另一个工作寄存器，一般称它为数据块地址指针。每做一次加法之后，修改地址指针，以便取出下一个数来相加，并且使计数器减 1。到计数器减到 0 时，求和结束，把和存入 SUM 即可。程序流程图如图 4 - 9 所示。

假设数据块长度存在内存单元 20H 中，数据块起始地址为 22H，结果存入内存单元 21H 中。

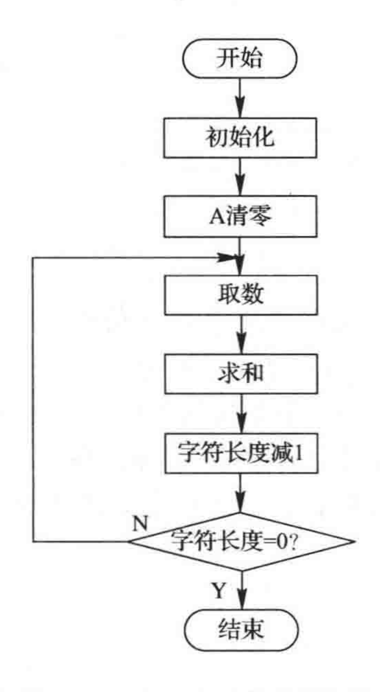


图 4 - 9　例 4 - 5 的程序流程图

参考程序 1：

```
LEN      DATA   20H          ；伪指令指定一个内存单元
SUM      DATA   21H          ；同上
BLOCK    DATA   22H          ；同上
         ORG    0000H
         CLR    A            ；清累加器
         MOV    R2，LEN      ；数据块长度送 R2
         MOV    R1，#BLOCK   ；数据块首地址送 R1
LOOP：   ADD    A，@R1       ；循环做加法
         INC    R1           ；修改地址指针，指向下一存储单元
         DJNZ   R2，LOOP     ；修改计数器并判断
         MOV    SUM，A       ；存和
         END
```

以上程序在计数器初值不为 0 时是没有问题的，但若数据块的长度为 0，则将出现问题。若 R2 初值为 0，减 1 之后将为 FFH，故要做 256 次加法之后才会停止，显然和题意不

符。若考虑到这种情况，则可按图 4－8(b)的方式来编写程序。在做加法之前，先判断一次 R2 的初值是否为 0。整个程序仍基本套用原来的形式。

参考程序 2：

```
        CLR   A           ；清累加器
        MOV   R2，LEN      ；数据块长度送 R2
        MOV   R1，#BLOCK   ；数据块首地址送 R1
        INC   R2           ；因为循环内需要先减“1”，所以要比循环次数多 1 次
NEXT：  DJNZ  R2，LOOP
        LJMP  CHECK
LOOP：  ADD   A，@R1       ；循环做加法
        INC   R1           ；修改地址指针，指向下一存储单元
        LJMP  NEXT         ；转向下一个数
CHECK： MOV   SUM，A       ；存和
```

【例 4－6】 将内部 RAM 中起始地址为 data 的数据串送到外部 RAM 中起始地址为 buffer 的存储区域中，直到发现“＄”字符，传送停止。

分析：这是一个不定循环次数的数据转移程序。循环次数事先不知道，需要先判断，后执行。以特殊字符“＄”作为结束，编程时，在循环体内部开始就进行特殊字符判断，遇到后立即转出循环；否则，完成操作后，进行无条件转移，继续循环。程序流程图如图 4－10所示。

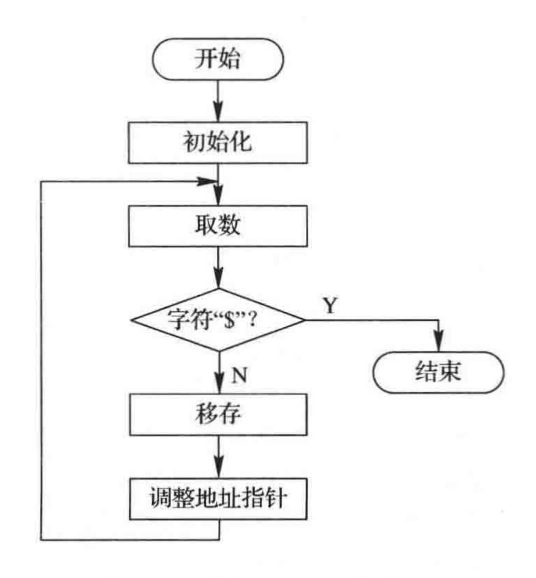


图 4－10　例 4－6 的程序流程图

参考程序：

```
    ORG   0000H
    MOV   R0，#data       ；data 是内部存储单元起始地址，在此没有定义具体值
    MOV   DPTR，#buffer   ；buffer 是外部存储单元起始地址，在此没有定义具体值
L1：MOV   A，@R0          ；读内部存储单元的数
```

```
       CJNE    A, #24H, L2        ; 判断是否为"$"字符，也可以用"CJNE  A, '$', L2"
       LJMP    L3                 ; 是，转结束
L2:    MOVX    @DPTR, A           ; 不是，传送数据
       INC     R0                 ; 指向下一个内部存储单元
       INC     DPTR               ; 指向下一个外部存储单元
       LJMP    L1                 ; 传送下一数据
L3:    NOP
       END
```

【例 4-7】 把例 4-1 三字节无符号数相加的程序改写成循环方式。

分析：针对例 4-1 的"参考程序 2"，就可以把程序改成循环方式。因为是三个字节相加，故循环次数为 3；因为循环体内统一用 ADDC，故要在循环前把进位位 C 置成"0"。用这种方法还可以实现更多位的加法或减法。

参考程序：

```
        ORG     0000H
        CLR     C               ; 进位位清 0
        MOV     R0, #52H        ; 被加数的低字节地址
        MOV     R1, #55H        ; 加数的低字节地址
        MOV     R7, #3          ; 置循环次数
LOOP:   MOV     A, @R0          ; 读被加数
        ADDC    A, @R1          ; 字节相加
        MOV     @R0, A          ; 存字节相加结果
        DEC     R0              ; 指向下一个被加数
        DEC     R1              ; 指向下一个加数
        DJNZ    R7, LOOP        ; 循环次数减"1"，不等于"0"则转到 LOOP
        MOV     00H, C          ; 进位送 00H 位保存
        END
```

4.2.4　子程序

在程序设计中，经常会遇到通用问题，同一个程序的不同地方要求执行同样的操作或运算，例如求各种函数和加减乘除运算、代码转换以及延时程序等，通常将这些能完成某种基本操作并具有相同操作的程序段单独编制成子程序，以供不同程序或同一程序反复调用。这样，一方面，程序结构简单，程序模块化、通用化，便于阅读；另一方面，节省了程序存储空间，还利于按照某种功能进行调试。在程序中需要执行这种操作的地方执行一条调用指令，转到子程序中完成规定操作，并返回原来程序中继续执行下去，这就是所谓的子程序结构。

调用子程序时应注意以下几点：

(1) 子程序调用由 LCALL 指令或 ACALL 指令产生，子程序中执行 RET 返回。

(2) 在子程序中，一般应包括现场保护和现场恢复，可通过堆栈实现保护和恢复现场。

(3) 如果需要从主程序传递参数到子程序，可以约定交换数据的地址单元、寄存器，或者采用堆栈方法。

(4) 一个子程序可以被另一个子程序调用，称之为子程序嵌套。

(5) 在子程序调用过程中，若使用堆栈，应注意给堆栈指针 SP 赋栈底初值，设置合适大小的栈区空间，也可以默认初始值，这时(SP)=07H。程序中使用堆栈指令时，PUSH 和 POP 一般要配对使用。

【例 4-8】 设 a、b 是个位数，分别存于内部 RAM 的 A1、A2 两个单元中，编程计算平方和 $c=a^2+b^2$，并将计算结果存入内部 RAM 的 A3 单元中。

分析： 考虑到 a、b 是个位数，我们可以设定一个个位数平方的表格，利用“MOVC A，@A+DPTR”语句，通过查表的方法来计算 a 和 b 的平方。因为 c 是两个数的平方和，所以两次求平方可以采用子程序。在子程序中用累加器 A 来传递参数。

参考程序：

```
        ORG     0000H
A1      EQU     30H             ；伪指令赋值，对应 a
A2      EQU     31H             ；伪指令赋值，对应 b
A3      EQU     32H             ；伪指令赋值，对应 c
        MOV     A，A1           ；取 a
        LCALL   SQR             ；调用查表子程序
        MOV     R1，A           ；a 的平方暂存 R1 中
        MOV     A，A2           ；取 b
        LCALL   SQR             ；调用查表子程序
        ADD     A，R1           ；求出平方和暂存在 A 中
        MOV     A3，A           ；结果存于 A3 中
        LJMP    $
SQR：   MOV     DPTR，#TAB      ；子程序
        MOVC    A，@A+DPTR
        RET
TAB：   DB 0，1，4，9，16，25，36，49，64，81
        END
```

【例 4-9】 将内部 RAM 从 30H 开始的 8 个单元的十六进制数转换成 ASCII 码，并存在内部 RAM 从 40H 开始的单元中，转换结果的高位存在前，低位存在后。

分析： 因为 8 个单元中，每个字节有两个十六进制数，可以编写单一字节的转换子程序，然后循环调用 8 次。转换子程序在例 4-10 中已编好，待转换数需传入 A 中，直接调用子程序 HEXASC，转换完成的结果：高位 ASCII 码在 B 中，低位 ASCII 码在 A 中。我们可以根据约定好的入口和出口参数直接调用。

参考程序：

```
        ORG     0000H
        MOV     R0，#30H        ；十六进制数存储单元地址指针
        MOV     R1，#40H        ；ASCII 码存储单元地址指针
        MOV     R7，#08H        ；循环次数
```

```
LOOP:  MOV     A, @R0        ；将十六进制数给 A，传入口参数
       LCALL   HEXASC        ；调用转换子程序
       XCH     A, B          ；将原来结果(出口参数)中的高位交换到 A 中
       MOV     @R1, A        ；存高位 ASCII 码，A 的内容存入间接寻址的单元
       INC     R1            ；指向下一个单元
       XCH     A, B          ；将原来结果(出口参数)中的低位交换到 A 中
       MOV     @R1, A        ；存低位 ASCII 码，A 的内容存入间接寻址的单元
       INC     R1            ；存数指向下一个单元
       INC     R0            ；取数指向下一个单元
       DJNZ    R7, LOOP      ；循环减 1，不为 0 则重复这一过程
       LJMP    $
       END
```

4.3　常用程序设计举例

本节设计了一些在单片机应用系统中的常用程序，并提供了多种编程思路和参考程序，这些程序均以子程序的形式给出。读者既可以学习汇编语言编程方法(包括子程序的标准结构)，也可以在将来的应用中直接调用这些子程序。子程序给定了入口名称和入口参数，运算结果在出口参数里。因为已规定了程序入口标号，任务也是子程序结构，最后的结束语句是 RET，所以不再用 ORG 规定程序的起始地址。

在子程序的设计过程中，需要考虑保护所有用到的 SFR、工作寄存器和内存单元，除了入口参数和出口参数所涉及的以外，有的是隐含在程序里的，比如：指令若影响了进位 C，就需要保护程序状态字 PSW 。

4.3.1　数制转换子程序

单片机能识别和处理的是二进制码，而输入/输出设备(如：LED 显示器、微型打印机等)则常使用 ASCII 码或十六进制数。为此，在单片机应用系统中经常需要通过程序进行二进制与 BCD 码或 ASCII 码的相互转换。

【例 4-10】 编写将累加器 A 中存放的十六进制数转换成两位 ASCII 码的子程序。入口名称：HEXASC。入口参数：A＝二位十六进制数。出口参数：B＝高位 ASCII 码，A＝低位 ASCII 码。

分析 1： 此例适用嵌套子程序，先编写一个低四位转换的子程序 HEASC，再由 HEXASC子程序调用。

对于十六进制数 0～9，其 ASCII 码是 30H～39H；对于十六进制数 A～F，其 ASCII 码是 41H～46H。可以利用“MOVC　A，@A＋DPTR”指令、查表方法，编写子程序。这种方法更容易理解。

程序中的 NOP 语句主要用于功能模块的分割，没有其他意义。

参考程序 1：

```
HEXASC: PUSH   ACC           ；原数保存到堆栈
        SWAP   A             ；先转换高四位，把高四位移到低四位
        LCALL  HEASC         ；转换高四位
        MOV    B, A          ；存放高四位的 ASCII 码
        POP    ACC           ；从堆栈再取原数
        LCALL  HEASC         ；转换低四位
        RET
HEASC:  NOP                  ；将累加器的低四位转换成 ASCII 码子程序
        ANL    A, #0FH       ；屏蔽低四位
        MOV    DPTR, #TAB    ；取表的首地址
        MOVC   A, @A+DPTR    ；查表
        RET                  ；子程序返回
TAB:    DB 30H, 31H, 32H, 33H, 34H, 35H, 36H, 37H, 38H, 39H    ；0~9 的 ASCII 码
        DB 41H, 42H, 43H, 44H, 45H, 46H    ；A~F 的 ASCII 码
```

分析 2： 通过判断十六进制数是 0～9 还是 A～F 来进行转换。判断方法最好用指令 CJNE，然后分别加上 30H 或 37H，编写子程序。也可以用减法指令 SUBB，由于减法指令得到的是差值，原来的数据被破坏，所以必须预先加以保护。

参考程序 2：

```
HEASC: NOP                   ；将累加器的低四位转换成 ASCII 码子程序
       PUSH  PSW             ；保护程序状态字
       ANL   A, #0FH         ；屏蔽低四位
       CJNE  A, #0AH, HE10   ；利用 CJNE 语句的减法功能，同时保持原数不变
HE10:  JC    HE20            ；有借位即是 0~9，否则为 A~F
       ADD   A, #07H         ；A~F 的 ASCII 码先加 07H，下面再加 30H，合计 37H
HE20:  ADD   A, #30H         ；0~9 的 ASCII 码只加 30H
       POP   PSW             ；恢复程序状态字
       RET
```

分析 3： 利用“DA　A”指令进行转换。

对于 0～9，其 BCD 码是＃00H～＃09H，加上 BCD 码＃90H，则分别是 BCD 码＃90H～＃99H。执行“DA　A”指令后，原数不变，再加 BCD 码＃40H，则应等于＃0C0H～＃0C9H；再执行“DA　A”指令后，变为＃30H～＃39H，恰好是 0～9 的 ASCII 码。

对于 A～F，其 BCD 码是＃10H～＃15H，加上 BCD 码＃90H，则分别是 BCD 码＃0A0H～＃0A5H，进位位为 1。执行“DA　A”指令后，变为＃00H～＃05H，再加 BCD 码＃40H 和进位位，则应等于＃41H～＃46H；再执行“DA　A”指令后，仍为＃41H～＃46H，恰好是 A～F 的 ASCII 码。

参考程序 3：

```
HEASC: NOP                   ；将累加器的低四位转换成 ASCII 码子程序
       ANL      A, #0FH      ；屏蔽低四位
```

```
        ADD     A，#90H         ；以下四条指令进行转换
        DA      A
        ADDC    A，#40H
        DA      A               ；转换完成
        RET
```

【例 4-11】 编写将单字节二进制数转换为 BCD 码的子程序。入口名称：BBCD。入口参数：A＝待转换 8 位二进制数。出口参数：B＝百位 BCD 码，A＝ 十位、个位 BCD 码。

分析 1：由于单字节二进制数最大是 255，所以最多是三位 BCD 码，可以采用除法指令编程，即先除以 100，余数再除以 10。

参考程序 1：

```
BBCD：MOV     B，#64H     ；将 100 装入寄存器 B 中
      DIV     AB          ；除以 100，商数是百位数进入 A 中，小于 100 的余数在 B 中
      PUSH    ACC         ；堆栈保存百位数
      MOV     A，#0AH     ；将 10 装入 A 中
      XCH     A，B        ；交换后，A 中是待转换数，B 中是"10"
      DIV     AB          ；除以 10，A 中商数是十位数，B 中余数是个位数
      SWAP    A           ；十位数放在高半字节
      ADD     A，B        ；个位数放在低半字节
      POP     B           ；将堆栈里的百位数推入到 B 寄存器中
      RET
```

分析 2：采用减法方法，先减 100，判断是否有借位。没有借位时，B＋1，循环；有借位时，多减了 100，需再加 100。其余类似。

参考程序 2：

```
BBCD：PUSH    PSW         ；用到了进位位，需保护程序状态字 PSW
      MOV     B，#00H     ；百位计数单元清 0
      CLR     C           ；进位位清 0
BB10：SUBB    A，#64H     ；－100
      JC      BB20        ；有借位转走
      INC     B           ；百位计数单元＋1
      LJMP    BB10        ；循环
BB20：ADD     A，#64H     ；把多减的 100 加回来
      PUSH    B           ；保存百位数
      MOV     B，#00H     ；十位计数单元清 0
      CLR     C           ；进位位清 0
BB30：SUBB    A，#0AH     ；－10
      JC      BB40        ；有借位转走
      INC     B           ；十位计数单元＋1
      LJMP    BB30        ；循环
BB40：ADD     A，#0AH     ；把多减的 10 加回来，A 中是个位数，B 中是十位数
      XCH     A，B        ；A 和 B 交换，A 中是十位数，B 中是个位数
      SWAP    A           ；十位数放在高半字节
      ADD     A，B        ；个位数放在低半字节
      POP     B           ；将堆栈里的百位数推入到 B 寄存器中
```

```
        POP     PSW             ；恢复程序状态字 PSW
        RET
```

4.3.2 延时子程序

单片机工作时有时需要一些时间的延时，但不用十分精确。比如：控制一个指示灯的亮灭用于提示，或 A/D 转换器启动后的等待时间等。这种情况下，用指令运行时间的累积来做延时比较便捷。

【例 4-12】 已知单片机的 $f_{osc}=12MHz$，设计延时 1ms 的子程序。入口名称：DELAY。入口参数：无。出口参数：无。

分析： 单片机指令在执行时需要一定的时间，这样我们就可以利用运行若干条指令的方式(一般是循环执行)，来获取一段延时，但这样的延时准确性和灵活性稍差。我们可以利用单片机内部的定时器/计数器，来获取精确的时间，具体方法见第 5 章的有关内容。

这里采用运行指令延时的方法。晶振频率为 $f_{osc}=12$ MHz 时，一个机器周期为 $12/f_{osc}=1\mu s$，因此延时 1 ms 的子程序的执行时间是 1000 个机器周期(1000 μs)。

需要执行的时间为 $(2+1+(1+1+2)\times 248+2+2)\ \mu s=999\ \mu s\approx 1$ ms，循环次数为 248。

参考程序：

```
DELAY：PUSH    07H         ；R7 入栈保护，2 个机器周期指令
       MOV     R7，#248    ；延时 1ms 子程序入口，1 个机器周期指令
LOOP： NOP                 ；1 个机器周期指令
       NOP                 ；1 个机器周期指令
       DJNZ    R7，LOOP    ；2 个机器周期指令
       POP     07H         ；出栈，R7 恢复，2 个机器周期指令
       RET                 ；2 个机器周期指令
```

通常，为了简单起见，我们只考虑循环体内的执行时间：$(1+1+2)\times 250\approx 1$ ms，循环次数为 250。

4.3.3 均值滤波子程序

在应用系统中有时需要对数据进行采样(比如：测温)，为了克服偶然因素引起的波动，一般需要采集 N 个值，求算术平均，这就是算术平均值滤波法。

【例 4-13】 已知采样值为单字节，连续采样 n 次，编写对采样值进行算术平均值滤波的子程序。入口名称：FILTER。入口参数：R0=采样首地址，R7=采样次数 n(n 只能取 1、2、4、8、16)。出口参数：A=平均值。

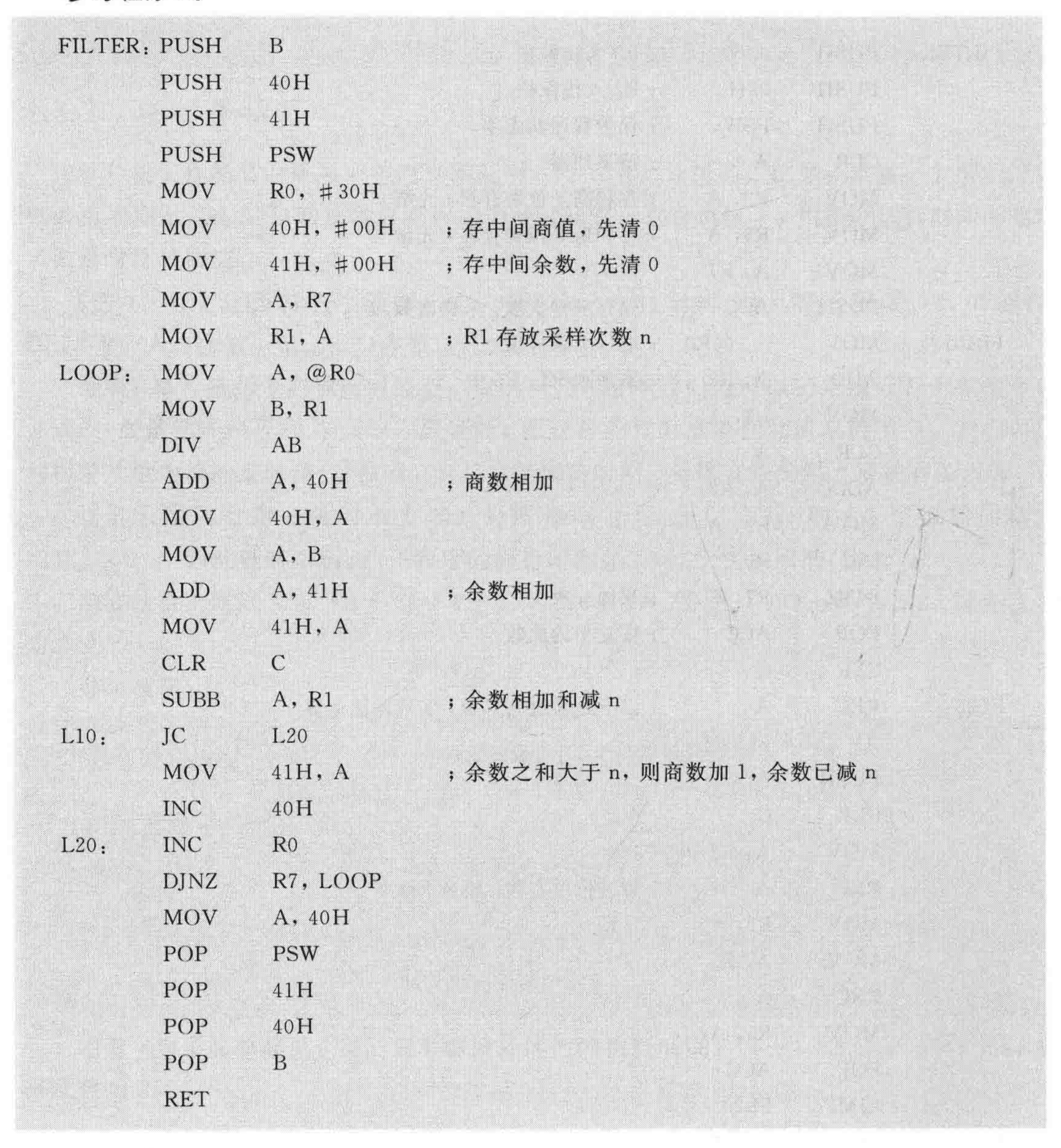
分析 1： 先实现 n 次数据求和，和的值可能会大于 255，因此需用两个存储器保存。然后再将和除以 n，由于 n 的取值是 1、2、4、8、16，为 2 的整数幂，因此通过右移位代替除法，可简化算法，提高运行效率。

参考程序 1：

```
FILTER:  PUSH    03H        ; R3 入栈保护
         PUSH    02H        ; R2 入栈保护
         PUSH    PSW        ; 保护程序状态字
         CLR     A          ; 清累加器
         MOV     R2, A      ; 存和高 8 位寄存器，先清 0
         MOV     R3, A      ; 存和低 8 位寄存器，先清 0
         MOV     A, R7
         PUSH    ACC        ; 保存采样次数，采样次数为 1、2、4、8、16
FL10:    MOV     A, @R0     ; 取一个采样值
         ADD     A, R3      ; 累加到 R2、R3 中
         MOV     R3, A
         CLR     A
         ADDC    A, R2
         MOV     R2, A
         INC     R0
         DJNZ    R7, FL10   ; 累加 n 次
         POP     ACC        ; 恢复平均次数
         CLR     C
FL20:    RRC     A          ; 以下是通过右移位，实现除法运算
         JC      FL30
         PUSH    ACC
         CLR     C
         MOV     A, R2
         RRC     A          ; 带进位位右移，相当于除 2
         MOV     R2, A
         MOV     A, R3
         RRC     A
         MOV     R3, A
         POP     ACC
         LJMP    FL20
FL30:    MOV     A, R3      ; 结果给 A
         POP     PSW        ; 程序状态字恢复
         POP     02H        ; R2 出栈恢复
         POP     03H        ; R3 出栈恢复
         RET
```

分析 2：当 n 为任意数时，如果还用先求和再除的方法，编程比较困难，因为汇编指令只提供了单字节除法。我们可以考虑采用先除以 n，再求和的方法。所有商数直接相加作为平均值，余数在相加过程中减 n，没有借位，则平均值+1，循环得到结果。也可以在余数相加过程中除以 n。

参考程序 2：



```
FILTER：PUSH    B
        PUSH    40H
        PUSH    41H
        PUSH    PSW
        MOV     R0，#30H
        MOV     40H，#00H      ；存中间商值，先清 0
        MOV     41H，#00H      ；存中间余数，先清 0
        MOV     A，R7
        MOV     R1，A          ；R1 存放采样次数 n
LOOP：  MOV     A，@R0
        MOV     B，R1
        DIV     AB
        ADD     A，40H         ；商数相加
        MOV     40H，A
        MOV     A，B
        ADD     A，41H         ；余数相加
        MOV     41H，A
        CLR     C
        SUBB    A，R1          ；余数相加和减 n
L10：   JC      L20
        MOV     41H，A         ；余数之和大于 n，则商数加 1，余数已减 n
        INC     40H
L20：   INC     R0
        DJNZ    R7，LOOP
        MOV     A，40H
        POP     PSW
        POP     41H
        POP     40H
        POP     B
        RET
```

4.3.4 数据极值查找子程序

【例 4-14】 编写外部存储器中最大数查找子程序。入口名称：MAXIM。入口参数：DPTR＝数据块首地址，R7＝数据块长度。出口参数：A＝最大数。

分析： 极值查找操作的主要内容是进行数值大小的比较。假定在比较过程中，以 B 单元存放比较出来的最大数，与之逐个比较的另一个数随时取出放在 A 中。比较结束后，把查找到的最大数放在 A 中。

采用循环程序结构，必须先设一个“0”，放到 B 中，数据区长度计数器是 R7，其他取数过程在循环体内。程序流程图如图 4-11 所示。

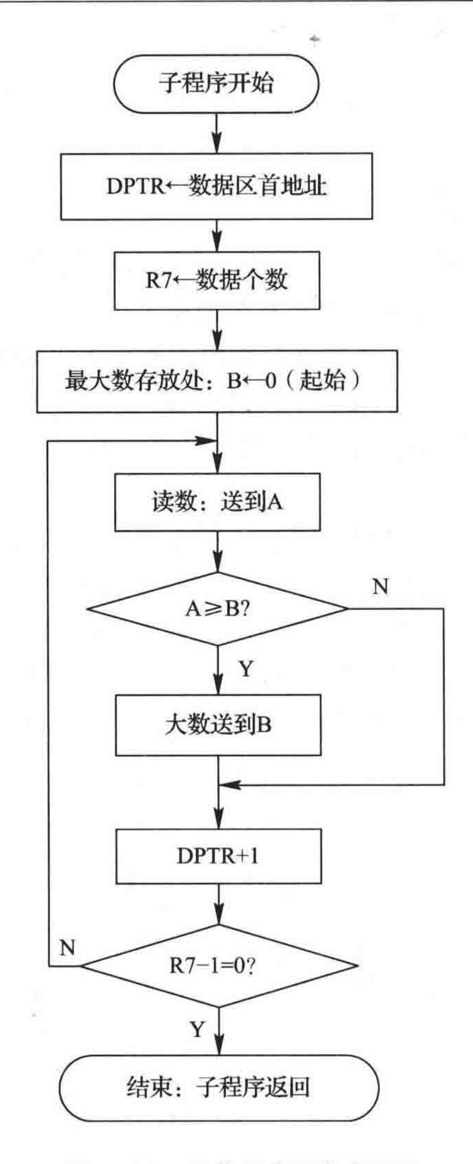


图 4-11　极值查找程序流程图

参考程序：

```
MAXIM：NOP                    ；大数查找子程序
       PUSH    PSW            ；保护程序状态字
       PUSH    B              ；保护寄存器 B
       MOV     B，# 00H       ；已选出的最大数存放处，开始设为"0"
LOOP：  MOVX    A，@DPTR       ；读下一个数
       CJNE    A，B，CHK      ；数值比较
CHK：   JC      NEXT           ；A 值"小"，则转移
       MOV     B，A           ；大数从 A 送到 B 寄存器
NEXT：  INC     DPTR
       DJNZ    R7，LOOP       ；继续
       MOV     A，B
       POP     B              ；恢复寄存器 B
```

```
        POP     PSW                 ；恢复程序状态字
        RET                         ；子程序返回
```

4.3.5 算术运算子程序

MCS-51 单片机指令集里只提供了简单的 8 位无符号数二进制加、减、乘、除指令，在实际的数据采集和控制系统中还需要更复杂的算术运算。

【例 4-15】 编写带符号双字节二进制数的加减法运算子程序。入口名称：BSUB 为减法程序入口，BADD 为加法程序入口。入口参数：R2R3=(被加数或被减数)，R4R5=(加数或减数)，其中 R2 和 R4 的最高位为两数的符号位。出口参数：R6R7=(和或差)，R6 的最高位是符号位。

分析：有符号数的加减运算比较复杂，首先把减法转换成加法，即如果是减法，就把减数的符号位取反；其次，判断两数的符号位，如果相同则作加法运算，如果相反则作减法运算；然后，在减法运算时，如果符号位为“0”则是正数，如果符号位为“1”则是负数，需进行转补码编程，即取反加“1”；最后，进行符号位处理。我们在高字节的最高位定义了符号位，如果在运算中占用了此位，则转到溢出处理。本例中不再考虑溢出处理部分。程序流程图如图 4-12 所示。

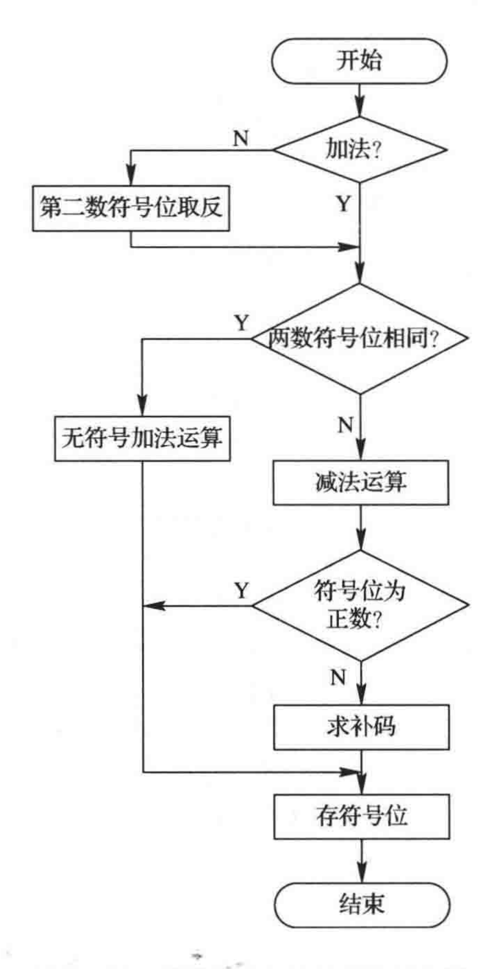


图 4-12 带符号位加减法程序流程图

参考程序：

```
BSUB：NOP                     ；减法入口
      PUSH   PSW              ；保护程序状态字
      MOV    A，R4            ；取减数高字节
      CPL    ACC.7            ；减数符号取反以进行加法运算
      MOV    R4，A
BADD：NOP                     ；加法入口
      MOV    A，R2            ；取被加数
      MOV    C，ACC.7
      MOV    F0，C            ；被加数符号保存在 F0 中
      XRL    A，R4            ；两数高字节异或
      MOV    C，ACC.7         ；两数同号 Cy=0，两数异号 Cy=1
      MOV    A，R2
      CLR    ACC.7            ；高字节符号位清 0
      MOV    R2，A            ；取其数值部分
      MOV    A，R4
      CLR    ACC.7            ；高字节符号位清 0
      MOV    R4，A            ；取其数值部分
      JC     JIAN             ；两数异号转 JIAN
JIA： MOV    A，R3            ；两数同号进行加法运算
      ADD    A，R5            ；低字节相加
      MOV    R7，A            ；保存和
      MOV    A，R2
      ADDC   A，R4            ；高字节相加
      MOV    R6，A            ；保存和
      JB     ACC.7，QAZ       ；符号位为"1"，转溢出处理
QWE： MOV    C，F0            ；结果符号处理
      MOV    ACC.7，C
      MOV    R6，A
      POP    PSW              ；恢复程序状态字
      RET                     ；子程序返回
JIAN：MOV    A，R3            ；两数异号进行减法运算
      CLR    C
      SUBB   A，R5            ；低字节相减
      MOV    R7，A            ；保存差
      MOV    A，R2
      SUBB   A，R4            ；高字节相减
      MOV    R6，A            ；保存差
      JNB    ACC.7，QWE       ；判断差的符号，为"0"转 QWE
BMP： MOV    A，R7            ；为"1"进行低字节取补
      CPL    A
```

```
        ADD    A，#1
        MOV    R7，A
        MOV    A，R6            ；高字节取补
        CPL    A
        ADDC   A，#0
        MOV    R6，A
        CPL    F0               ；保存在F0中的符号取反
        LJMP   QWE              ；转结果符号处理
   QAZ：NOP                     ；溢出处理
        ⋮
```

思考与练习

1. 伪指令的作用是什么？伪指令和指令用什么区分？END是单片机执行的结束符吗？

2. 汇编语言设计的主要步骤是什么？

3. 何谓程序框图？什么情况下需要画程序框图？

4. 常用的程序结构有哪几种？特点如何？

5. 何谓子程序？一般在什么情况下采用子程序？子程序的结构特点是什么？

6. 子程序调用和返回时，参数传递方法有哪几种？

7. 子程序调用堆栈起什么作用？

8. 试通过调用例4-14的数据极值查找子程序编程把片外RAM从2000H开始的连续50个单元的内容按降序排列，结果存入原存储区中。

9. 片内RAM首地址为40H，长度为32的数据区中，以ASCII码形式存放着一组字母，试编写程序，查找出X的个数，并将其存入地址为2010H的外存储单元中。

10. 片内存储器30H单元内存放一个个位整数，编程求其平方根(精确到5位有效数字)，将平方根存放到以50H为首地址的内存中。

11. 编写程序，要求将片外数据存储器地址为1000H～1030H区域的数据块，全部搬移到片内RAM从30H起始的存储区域，并将原数据区全部填为00H。

12. 试编程计算内部RAM 30H开始单元的10个数的平均值，结果存放在40H开始的2个单元中，可以调用例4-13的均值滤波子程序。

13. 在片外2000H开始的单元中有50个有符号数，试编写程序统计其中正数、负数和0的个数，并存放到30H、31H、32H中。

14. 分析以下程序的功能：

```
LOAD：MOV     R1，#40H
      MOV     R7，#10H
      MOV     DPTR，#3000H
NEXT：MOV     A，@R1
```

```
MOVX    @DPTR, A
INC     R1
INC     DPTR
DJNZ    R7, NEXT
LJMP    $
```

15. 在以 1000H 为首地址的存储区中，存放 10 个用 ASCII 码表示的 0～9 之间的数，试编程将它们转换成 BCD 码，并以压缩 BCD 码(即一个单元存放两位 BCD 码)的形式存放在 1500H～1504H 单元中。

16. 改写例 4-14 的子程序，使之变成一个内部数据存储器中最小数查找子程序。

17. 在外部数据存储器的 1000H～100FH 单元中共有 16 个数，试利用例 4-14 的子程序，编写一个从大到小的排序程序，排序结果存入 1100H～110FH 单元中。

18. 在外部数据存储器的 1000H～100FH 单元中共有 16 个数，试利用例 4-14 的子程序(可以改写子程序的少量代码)，编写一个从小到大的排序程序，排序结果存入 1100H～110FH 单元中。

19. 设 a、b 是个位数，分别存于内部 RAM 的 A1、A2 两个单元中，编程计算平方差 $c=a^2-b^2$，并将计算结果存入内部 RAM 的 A3 单元中。

第5章 单片机中断和定时器/计数器

MCS-51系列单片机得到广泛应用的一个重要原因是在一个芯片内集成了应用系统所需的大部分硬件功能，而这些硬件功能是由内部标准功能单元实现的。内部标准功能单元主要包括中断系统、定时器/计数器和串行通信接口。

本章从应用的角度重点讲述MCS-51单片机的中断系统和定时器/计数器的结构、工作原理、实现过程和汇编语言编程方法。

5.1 单片机的中断系统

中断系统在计算机系统中起着十分重要的作用，一个功能很强的中断系统，能大大提高计算机对外部事件的处理效率，增强实时性。MCS-51单片机具备一套完善的中断系统，包括5个中断源、2个中断优先级，可以实现两级中断嵌套，开发者通过软件可实现对中断的控制。

5.1.1 中断系统的基本概念和基本结构

1. 中断的基本概念

在日常生活中，有很多中断的例子。例如一个人正在家中看书，突然门铃或电话铃响了(必须马上处理)，他必须暂时停止看书，做好记号，去开门或者接电话，待事情处理完毕后，按照标记的记号再继续之前的阅读工作。这种正常的工作过程被外部事件打断的现象就是中断。

在单片机中，当CPU正在处理某件事情的时候(如正在执行主程序时)，单片机外部或内部发生的某一事件(如某个引脚上电平的变化，一个脉冲沿的发生或计数器的计数溢出等)请求CPU迅速去处理，于是CPU暂时中止当前的工作，转去处理所发生的事件，事件处理完毕后，CPU再回到刚刚被暂停的地方继续原来的工作，这个过程称为中断，如图5-1所示。

能引起CPU产生中断的事件，称为中断源。中断源向CPU提出的处理请求，称为中断请求。CPU接受中断请求，暂时中止自身的事情转去处理事件的过程，称为中断响应过程。CPU对事件的整个处理过程，称为中断服务。为实现中断而编写的服务程序，称为中断服务程序。事件处理完毕，再回到原来被中断的地方(即断点)，称为中断返回。单片机是通过相应的硬件电路和软件程序来完成中断功能的，所以将能完成中断功能的硬件系统和软件系统统称为中断系统。

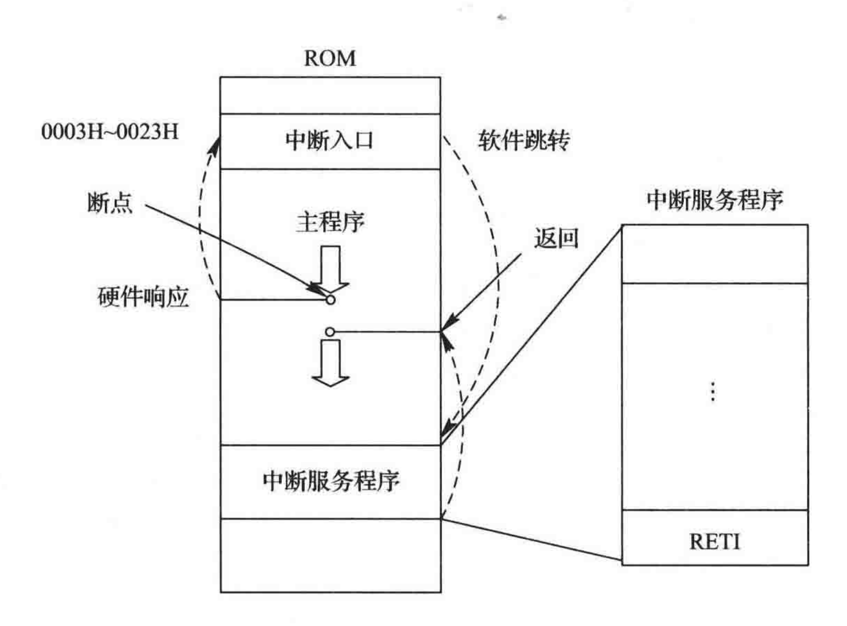


图 5-1　中断过程流程图

中断是 MCS-51 单片机的一个重要功能。采用中断技术可以使单片机实现以下功能：

(1) 分时操作。单片机的中断系统可以使 CPU 与外设同步工作。CPU 在启动外设后，可以继续执行主程序。而外设把数据准备好后，发出中断请求，CPU 响应该中断请求并为其服务。中断处理完成后，CPU 恢复执行主程序，外设也继续工作。因此，CPU 可以指挥多个外设同时工作，从而大大提高了 CPU 的利用率和输入/输出的速度。

(2) 实时处理。当单片机用于实时控制时，现场采集到的各种参数、信息随时可能发出中断请求，若中断是开放的，CPU 就可以立即响应并加以处理。

(3) 故障处理。如果单片机在运行过程中出现了事先预料不到的情况或故障(如掉电、存储器奇偶校验出错、运算溢出等)，可以利用中断系统自行处理而不必停机。

2. 中断系统的基本结构

MCS-51 单片机的中断系统包括 5 个中断源、2 个中断优先级(可以实现两级中断嵌套)、4 个用于中断控制的寄存器(IE、IP、TCON 和 SCON)，其中断系统结构如图 5-2 所示。

MCS-51 单片机的 5 个中断源分别为：

(1) $\overline{\text{INT0}}$：外部中断 0 请求，可由 IT0 选择其有效方式。当 CPU 检测到$\overline{\text{INT0}}$(P3.2)引脚上出现有效的中断信号时，置位 IE0，向 CPU 申请中断。

(2) $\overline{\text{INT1}}$：外部中断 1 请求，可由 IT1 选择其有效方式。当 CPU 检测到$\overline{\text{INT1}}$(P3.3)引脚上出现有效的中断信号时，置位 IE1，向 CPU 申请中断。

(3) T0：定时器/计数器 T0 溢出中断请求。当定时器/计数器 T0 发生溢出时，置位 TF0，向 CPU 申请中断。

(4) T1：定时器/计数器 T1 溢出中断请求。当定时器/计数器 T1 发生溢出时，置位 TF1，向 CPU 申请中断。

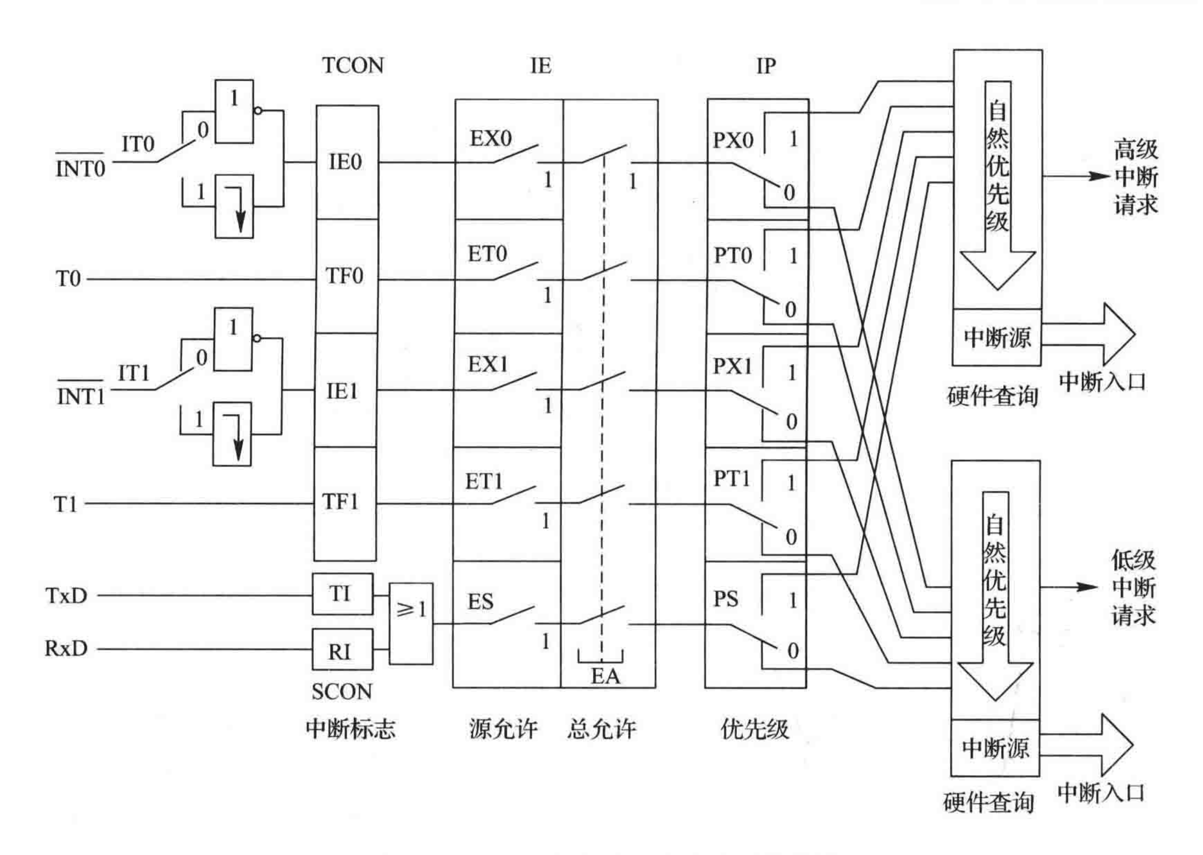


图 5-2　MCS-51 单片机的中断系统结构

(5) TxD/RxD：串行口中断请求。当串行口接收完一帧串行数据时置位 RI，或当串行口发送完一帧串行数据时置位 TI，向 CPU 申请中断。

这些中断源的中断请求标志位分别设置在特殊功能寄存器 TCON 和 SCON 中。

通常，外部中断源有以下几种：

(1) I/O 设备中断源。键盘、打印机、A/D 转换器等 I/O 设备在完成自身的操作后，向单片机发出中断请求，请求单片机对其进行处理。

(2) 控制对象中断源。当单片机用作实时控制时，被控对象常常被用作中断源，用于产生中断请求信号，要求单片机及时采集系统的控制参量、越限参数以及要求发送和接收数据等。例如，电压、电流、温度、压力、流量和流速等超越上限和下限以及开关和继电器的闭合或断开都可以作为中断源来产生中断请求信号，并要求单片机通过执行中断服务程序来加以处理。

(3) 故障中断源。单片机外部故障源引起外部中断，如掉电中断等。在掉电时，掉电检测电路检测到它时就自动产生一个掉电中断请求，单片机检测到后便可以在大滤波电容维持正常供电的几秒内通过执行掉电中断服务程序来保护现场和启用备用电池，以便电源恢复正常后继续执行掉电前的用户程序。

5.1.2　中断系统的控制与实现

MCS-51 单片机对中断的控制是通过 4 个特殊功能寄存器实现的，它们分别是中断请求标志寄存器 TCON、串行口控制寄存器 SCON、中断允许控制寄存器 IE 和中断优先级

控制寄存器IP。

1. 中断请求控制

1）TCON中的中断标志位

TCON是定时器/计数器T0和T1的控制寄存器，同时也锁存T0和T1的溢出中断请求标志及外部中断0和外部中断1的中断请求标志等。TCON是可位寻址的特殊功能寄存器，寄存器字节地址为88H，位地址为88H～8FH。TCON的格式如下：

	D7	D6	D5	D4	D3	D2	D1	D0
(88H)	8FH	8EH	8DH	8CH	8BH	8AH	89H	88H
TCON	TF1	TR1	TF0	TR0	IE1	IT1	IE0	IT0

与中断有关的标志位有6个，具体含义如下：

① IT0：外部中断0触发方式控制位。该位由软件置1或清0。

当IT0=0时，外部中断0为电平触发方式。CPU在每个机器周期的S5P2时刻对$\overline{INT0}$(P3.2)引脚的电平进行采样，若采样到低电平，则表示中断信号有效。

电平触发方式时，外部中断源的有效低电平必须保持到请求获得响应时为止，否则就会漏掉；在中断服务结束之前，中断源的有效低电平必须撤除，否则中断返回之后将再次产生中断。该方式适合于外部中断输入为低电平，且在中断服务程序中能清除外部中断请求源的情况。如并行接口芯片8255的中断请求线在接受读或写操作后即被复位，因此以其去请求电平触发方式的中断比较方便。

当IT0=1时，外部中断0为边沿触发方式，CPU在每个机器周期的S5P2时刻对$\overline{INT0}$(P3.2)引脚的电平进行采样，如果在连续的两个机器周期检测到$\overline{INT0}$引脚由高电平变为低电平，则表示中断信号有效。

边沿触发方式时，在相继两次采样中，先采样到外部中断输入为高电平，下一个周期采样到外部中断输入为低电平，则置位中断申请标志IE0。若CPU暂时不能响应，中断申请标志也不会丢失，直到CPU响应此中断时才清0。另外，为了保证下降沿能够被可靠地采样到，$\overline{INT0}$引脚上的负脉冲宽度至少要保持一个机器周期。边沿触发方式适合于以负脉冲形式输入的外部中断请求，如ADC0809的转换结束信号EOC为正脉冲，经反相后就可以作为MCS-51的外部中断请求信号。

② IE0：外部中断0的中断请求标志位。当CPU检测到$\overline{INT0}$引脚上出现有效的中断信号时(若IT0=0，且检测到$\overline{INT0}$引脚为低电平时；若IT0=1，且检测到$\overline{INT0}$引脚出现负跳变时)，IE0由硬件置1，向CPU申请中断。

③ IT1：外部中断1触发方式控制位。其功能与IT0类似。

④ IE1：外部中断1的中断请求标志位。其功能与IE0类似。

⑤ TF0：T0溢出中断请求标志位。当启动定时器/计数器T0计数后，T0从初值开始加1计数，当最高位产生溢出时，TF0由硬件置1，向CPU申请中断。

CPU响应TF0中断时，由硬件清0TF0。

⑥ TF1：T1溢出中断请求标志位。其功能与TF0类似。

2）SCON中的中断标志位

SCON是串行口控制寄存器，也是可位寻址的特殊功能寄存器，寄存器字节地址为

98H，位地址为 98H～9FH。SCON 的格式如下：

	D7	D6	D5	D4	D3	D2	D1	D0
(98H)	9FH	9EH	9DH	9CH	9BH	9AH	99H	98H
SCON	SM0	SM1	SM2	REN	TB8	RB8	TI	RI

与中断有关的标志位有两个，具体含义如下：

① TI：串行口发送中断请求标志位。每当串行口发送完一帧串行数据后，TI 由硬件自动置 1。CPU 响应该中断时，不能自动清除 TI，必须在中断服务程序中用软件对 TI 标志位清 0。

② RI：串行口接收中断请求标志位。每当串行口接收完一帧串行数据后，RI 由硬件自动置 1。CPU 响应该中断时，不能自动清除 RI，必须在中断服务程序中用软件对 RI 标志位清 0。

其他位的功能参见 6.2.3 小节。

2. 中断允许控制

MCS - 51 单片机对中断源的开放或屏蔽，是由中断允许寄存器 IE 控制的。IE 是可位寻址的特殊功能寄存器，寄存器字节地址为 A8H，位地址为 A8H～AFH。IE 格式如下：

	D7	D6	D5	D4	D3	D2	D1	D0
(A8H)	AFH	AEH	ADH	ACH	ABH	AAH	A9H	A8H
IE	EA			ES	ET1	EX1	ET0	EX0

IE 中各位的含义如下：

① EA：中断允许总控制位。若 EA＝0，则所有中断请求被屏蔽(CPU 关中断)；若 EA＝1，则 CPU 开放所有中断(CPU 开中断)，但 5 个中断源的中断请求是否被允许，还要由 IE 中的低 5 位所对应的 5 个中断请求允许控制位的状态来决定，即 IE 对中断的开放和关闭为两级控制。

② ES：串行中断允许位。若 ES＝0，则禁止串行口中断；若 ES＝1，则允许串行口中断。

③ ET1：定时器/计数器 T1 的溢出中断允许位。若 ET1＝0，则禁止 T1 溢出中断；若 ET1＝1，则允许 T1 溢出中断。

④ EX1：外部中断 1 中断允许位。若 EX1＝0，则禁止外部中断 1 中断；若 EX1＝1，则允许外部中断 1 中断。

⑤ ET0：定时器/计数器 T0 的溢出中断允许位。若 ET0＝0，则禁止 T0 溢出中断；若 ET0＝1，则允许 T0 溢出中断。

⑥ EX0：外部中断 0 中断允许位。若 EX0＝0，则禁止外部中断 0 中断；若 EX0＝1，则允许外部中断 0 中断。

MCS - 51 单片机复位后，IE 各位被复位成"0"状态，所有中断请求被禁止，用户必须通过主程序中的指令来开放所需中断，以便相应中断请求来到时为 CPU 所响应。对 IE 各位的操作，既可以用字节操作指令来编写，也可以由位操作指令来实现。

【例 5 - 1】 假设允许外部中断 0 和定时器/计数器 0 中断，禁止其他中断源的中断请

求，试根据假设设置 IE 的相应值。

分析：本例的功能可以分别利用位操作指令和字节操作指令来实现。

参考程序 1(用位操作指令)：

```
CLR     ES      ；禁止串行口中断
CLR     EX1     ；禁止外部中断 1 中断
CLR     ET1     ；禁止定时器/计数器 T1 中断
SETB    ET0     ；允许定时器/计数器 T0 中断
SETB    EX0     ；允许外部中断 0 中断
SETB    EA      ；CPU 开中断
```

参考程序 2(用字节操作指令)：

```
MOV   IE，#10000011B    ；令 EA=1、ET0=1 和 EX0=1
```

或者用：

```
MOV   IE，#83H          ；用字节操作的方法直接写到寄存器 IE 中
```

或者用：

```
MOV   0A8H，#83H        ；用字节操作的方法直接写到地址 A8H 指向的寄存器 IE 中
```

3. 中断优先级控制

MCS-51 单片机的中断源有两个中断优先级，每一个中断源均可由软件设定为高优先级中断或低优先级中断，可实现两级中断嵌套，即当 CPU 正在处理一个中断请求时，又出现了另一个优先级比它高的中断请求，这时，CPU 就暂时中止原中断服务程序的执行，保护当前断点，转去响应优先级更高的中断请求。等到处理完更高级别的中断服务程序后，再继续执行原来较低级的中断服务程序。两级中断嵌套的过程如图 5-3 所示。

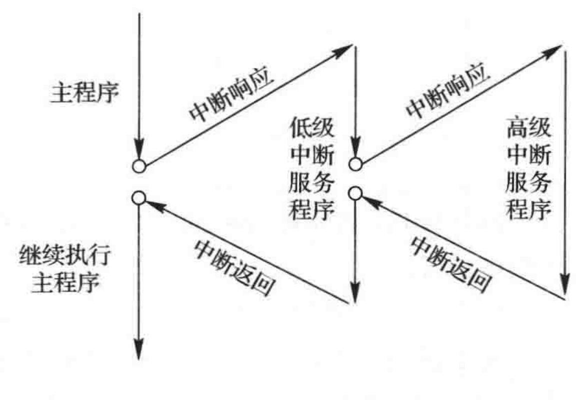


图 5-3　两级中断嵌套

由图 5-3 可见，一个正在执行的低优先级中断程序能被高优先级的中断源所中断，但不能被另一个低优先级的中断源所中断。若 CPU 正在执行高优先级的中断，则不能被任何中断源所中断。

MCS-51 单片机各中断源的优先级都是由中断优先级控制寄存器 IP 中的相应位来规定的。IP 是可位寻址的特殊功能寄存器，寄存器字节地址为 B8H，位地址为 B8H～BFH。IP 的格式如下：

	D7	D6	D5	D4	D3	D2	D1	D0
(B8H)	BFH	BEH	BDH	BCH	BBH	BAH	B9H	B8H
IP				PS	PT1	PX1	PT0	PX0

IP 中各位的含义如下：

① PX0：外部中断 0 中断优先级设定控制位。若 PX0＝1，则外部中断 0 被设定为高优先级中断；若 PX0＝0，则外部中断 0 被设定为低优先级中断。

② PT0：定时器/计数器 0 中断优先级设定控制位。若 PT0＝1，则 T0 被设定为高优先级中断；若 PT0＝0，则 T0 被设定为低优先级中断。

③ PX1：外部中断 1 中断优先级设定控制位。若 PX1＝1，则外部中断 1 被设定为高优先级中断；若 PX1＝0，则外部中断 1 被设定为低优先级中断。

④ PT1：定时器/计数器 1 中断优先级设定控制位。若 PT1＝1，则 T1 被设定为高优先级中断；若 PT1＝0，则 T1 被设定为低优先级中断。

⑤ PS：串行口中断优先级设定控制位。若 PS＝1，则串行口被设定为高优先级中断；若 PS＝0，则串行口被设定为低优先级中断。

MCS－51 单片机复位后，IP 被全部清 0，即全部设置为低优先级中断。当同时接收到多个同优先级的中断请求时，响应哪个中断源取决于内部硬件的查询顺序。同级中断源的优先级顺序如表 5－1 所示。

表 5－1　同级中断源的优先级顺序

中断源	同级的中断优先级
$\overline{INT0}$	最高
T0	↓
$\overline{INT1}$	↓
T1	↓
RxD/TxD	最低

由此可知，**各中断源在同一个优先级的条件下，外部中断 0 的优先权最高，串行口的优先权最低。**

MCS－51 的中断系统有两个不可寻址的“优先级激活触发器”：一个用来指示某高优先级的中断正在执行，所有后来的中断均被阻止；另一个用来指示某低优先级的中断正在执行，所有同级中断都被阻止，但不阻断高优先级的中断请求。

【例 5－2】 设 MCS－51 的外部中断 0 和定时器/计数器 0 中断为高优先级，其他中断请求为低优先级，试设置 IP 寄存器的相应值。

参考程序 1(用位操作指令)：

```
SETB    PX0
SETB    PT0
CLR     PX1
CLR     PT1
CLR     PS
```

参考程序 2(用字节操作指令)：

```
MOV     IP，#00000011B   ；令 PT0=1 和 PX0=1
```

或者用：

```
MOV     IP, #03H          ；用字节操作的方法直接写到寄存器 IP 中
```

或者用：

```
MOV     0B8H, #03H        ；用字节操作的方法直接写到地址 B8H 指向的寄存器 IP 中
```

5.1.3　中断系统的处理过程

中断处理过程可分为三个阶段，即中断响应、中断处理和中断返回。

1. 中断响应

1）中断响应条件

CPU 响应中断的条件如下：

（1）有中断源发出中断请求。

（2）中断总允许位 EA=1，即 CPU 开中断。

（3）申请中断的中断源的中断允许位为 1，即该中断没有被屏蔽。

（4）无同级或更高级中断正在被服务。

（5）当前指令已执行到最后一个机器周期，即在完成正在执行的指令前，不会响应中断，从而保证每条指令在执行过程中不被打断。

（6）若当前正在执行的指令是 RETI 或是访问 IE、IP 的指令，则在执行 RETI 或写入 IE、IP 指令之后，不能马上响应中断请求，至少再执行一条其他指令之后才会响应。

2）中断响应过程

MCS-51 单片机的 CPU 在每个机器周期的 S5P2 时刻顺序采样每个中断源的中断标志位，然后在下一个机器周期查询该采样。如果查询到某个中断标志为 1，则表明该中断源有中断请求。如果满足中断响应的条件，则在下一个机器周期开始时按优先级进行中断处理，并响应该中断。

MCS-51 单片机一旦响应中断，首先置位相应的中断“优先级激活触发器”，以阻止同级和低级中断，然后由硬件自动生成一条长调用指令“LCALL addr16”，把 PC(程序计数器)当前值压入堆栈，以保护断点，再将相应的中断入口地址送入 PC，使程序转向该中断入口地址单元中，以执行中断服务程序。各中断源的入口地址如表 5-2 所示。

表 5-2　中断源及其对应的入口地址

中断源	入口地址
$\overline{INT0}$	0003H
T0	000BH
$\overline{INT1}$	0013H
T1	001BH
RxD/TxD	0023H

由表 5-2 可知，MCS-51 单片机的两个相邻中断源的入口地址只相差 8 个字节，一般的中断服务程序都超过 8 个字节，所以通常是在中断入口地址单元放一条长转移指令

LJMP，这样可以使中断服务程序灵活地安排在 64KB ROM 中的任何地方，如图 5－1 所示。

中断服务程序从中断入口地址开始执行，一直到返回指令 RETI 为止，CPU 执行完这条指令后，把响应中断时所置位的优先级激活触发器清 0，然后从堆栈中弹出断点地址放入 PC，使 CPU 返回主程序。

3）中断响应时间

从查询中断请求标志位到转向中断服务入口地址所需的机器周期数称为中断响应时间。

根据中断响应条件，如果查询到中断请求信号的这个机器周期恰好处于正在执行指令的最后一个机器周期，那么在这个机器周期结束后，将由 CPU 自动执行一条硬件子程序调用指令 LCALL，以转到相应的中断入口地址单元，从而对中断进行处理。因为查询中断标志需要 1 个机器周期，长调用指令需要 2 个机器周期，所以中断响应时间最短为 3 个机器周期。

如果查询到中断请求信号时刚好是刚开始执行 RETI 或是访问 IE、IP 的指令，则需要把当前指令执行完再继续执行一条指令后，才能响应中断。执行上述的 RETI 或是访问 IE、IP 的指令最长需要 2 个机器周期，而接着再执行的一条指令，最长需要 4 个机器周期（乘法指令 MUL 或除法指令 DIV），再加上硬件子程序调用指令 LCALL 需要 2 个机器周期，所以中断响应时间最长为 8 个机器周期。

对于一个单中断系统，**中断响应时间总是在 3～8 个机器周期之间**。

4）中断请求的撤销

中断请求被响应后，要及时撤销中断请求，否则会引起重复响应。

（1）定时器/计数器中断请求的撤销。

中断请求被响应后，硬件自动将中断请求标志位 TF0 或 TF1 清 0，因此定时器/计数器中断请求是自动撤销的。

（2）外部中断请求的撤销。

外部中断请求的撤销包括中断标志位的清 0 和外部中断信号的撤销。

边沿触发方式：中断被响应后，IE0 或 IE1 由硬件自动清 0，由于负脉冲信号过后就消失了，所以外部中断请求也是自动撤销的。

电平触发方式：中断被响应后，IE0 或 IE1 由硬件自动清 0，但中断请求信号的低电平可能继续存在，在以后的机器采样中，又会把已清 0 的 IE0 或 IE1 重新置 1，造成重复中断。为了避免这种情况出现，应尽量采用边沿触发方式。

如果采用电平触发方式，要彻底撤销外部中断请求，需在中断响应后把中断请求信号引脚从低电平强制改变为高电平。因为 CPU 无法直接干预外电路，所以在引脚处用硬件电路（再配合相应的软件）来撤销外电路过期的中断请求。图 5－4 给出了一种低电平触发后的中断标志位撤销电路。

为实现图 5－4 所示电路的撤销中断请求功能，需要在中断服务程序中加如下两条指令：

```
SETB    P1.0 或 ORL    P1，#01H    ；将 P1.0 位置 1
CLR     P1.0 或 ANL    P1，#0FEH   ；再将 P1.0 位清 0
```

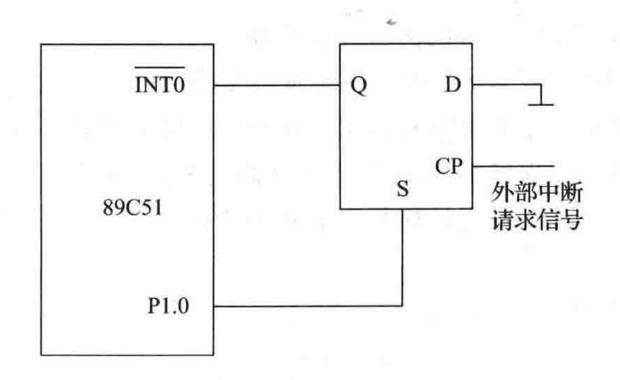


图 5-4　电平方式外部中断请求的撤销电路

第一条指令执行后，P1.0 位有一个“1”信号输出给 D 触发器的 S 端，将 Q 端直接置“1”(清除了标志位)；第二条指令执行后，P1.0 位恢复“0”，D 触发器的 Q 端将在 CP 的作用下由 D 端控制。

由上述分析可知，只要 P1.0 端输出一个负脉冲就可以使 D 触发器置“1”，从而撤销低电平的中断请求信号。

(3) 串行口中断请求的撤销。

串行口中断被响应后，CPU 无法知道是接收中断还是发送中断，还需测试这两个中断标志位的状态，以判定是接收操作还是发送操作。所以**串行口中断请求的撤销只能使用软件方法**，即在中断服务程序中用如下指令清除标志位：

```
CLR    TI    ；清 TI 标志位
CLR    RI    ；清 RI 标志位
```

2. 中断处理

CPU 响应中断后即转至中断服务程序的入口，执行中断服务程序。从中断服务程序的第一条指令开始到返回指令为止，这个过程称为中断处理或中断服务。不同的中断源服务的内容及要求各不相同，其处理过程也就有所区别。一般情况下，中断处理包括两部分内容：一是保护现场，二是为中断源服务。

所谓现场，是指中断时刻单片机中某些寄存器(状态寄存器 PSW、累加器 A、工作寄存器等)和存储器单元中的数据或状态。为了使中断服务程序的执行不破坏这些数据或状态，以免在中断返回后影响主程序的运行，要求把它们送入堆栈中保存起来，这就是现场保护。中断处理结束后，在返回主程序前，需要把保存的现场内容从堆栈中弹出，以恢复那些寄存器和存储器单元中的原有内容，这就是现场恢复。现场保护一定要位于中断处理程序的前面，而现场恢复一定要位于中断处理程序的后面。至于要保护哪些内容，应该由用户根据中断处理程序的具体情况来决定。

所谓中断源服务，是指根据中断源的具体要求进行相应的处理。

编写中断处理程序时应注意以下几点：

(1) 如果中断服务程序的长度超过 8 字节(即两个相邻中断源入口地址单元之间的距离)，则需在中断入口地址单元处放一条无条件转移指令 LJMP，使中断服务程序可灵活地安排在 64 KB ROM 中的任何空间。

(2) 若在执行当前中断程序时禁止更高优先级中断，则应用软件关闭 CPU 中断或屏

蔽更高级中断源的中断，在中断返回前再开放中断。

（3）在现场保护和现场恢复时，为了不使现场信息受到破坏或造成混乱，一般应关闭CPU中断，使CPU暂不响应新的中断请求。这样，在编写中断服务程序时，应注意在现场保护之前要关闭中断，在现场保护之后若允许高优先级中断嵌套，则应开中断。同样，在现场恢复之前应关闭中断，在现场恢复之后再开中断。在确定没有高级中断时，上述过程也可以简化。

中断处理流程图如图 5－5 所示。

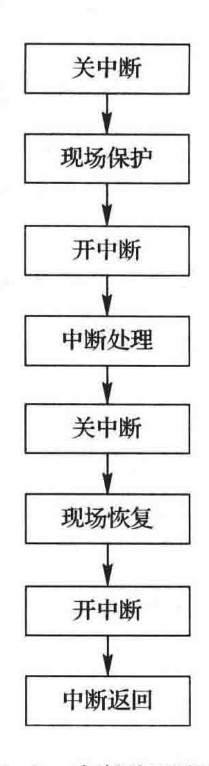
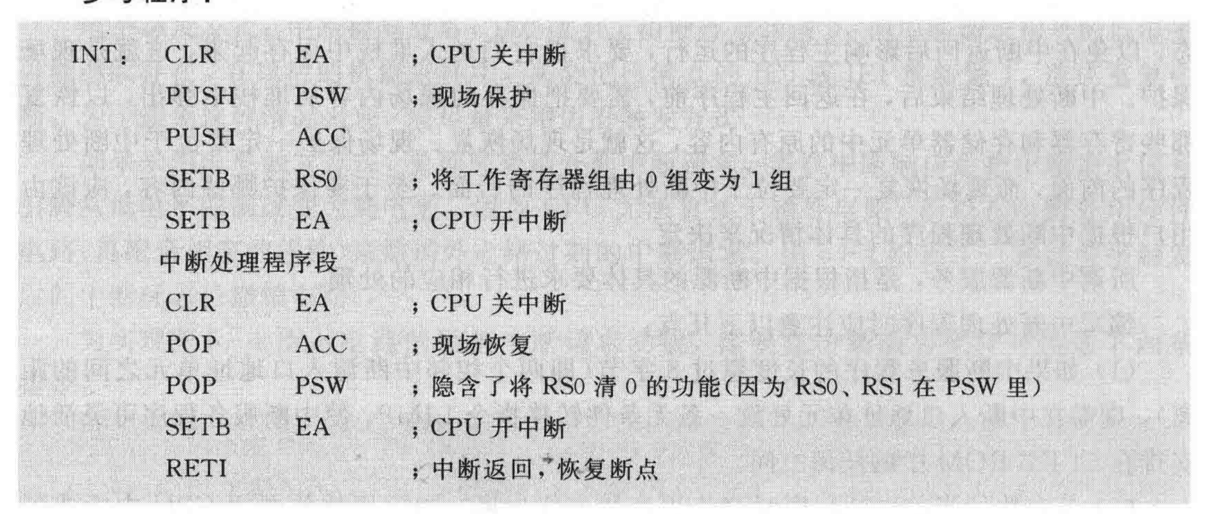


图 5－5　中断处理流程图

【例 5－3】 假设需要将 PSW、A 和工作寄存器 R0～R7(00H～07H)的内容进行现场保护，请根据中断处理流程图编写中断处理程序。

参考程序：



```
INT：  CLR     EA      ；CPU 关中断
       PUSH    PSW     ；现场保护
       PUSH    ACC
       SETB    RS0     ；将工作寄存器组由 0 组变为 1 组
       SETB    EA      ；CPU 开中断
      中断处理程序段
       CLR     EA      ；CPU 关中断
       POP     ACC     ；现场恢复
       POP     PSW     ；隐含了将 RS0 清 0 的功能(因为 RS0、RS1 在 PSW 里)
       SETB    EA      ；CPU 开中断
       RETI            ；中断返回，恢复断点
```

说明：

① 因为 Rn(n=0～7)的物理地址是由 PSW 中 RS1 和 RS0 的状态决定的，所以在对 PSW 进行保护之后，可以通过修改 RS1 或 RS0 的值实现对 Rn 的现场保护。

本例中的 R0～R7 属于第 0 组工作寄存器，对应地址为 00H～07H，若要采用压入堆栈的方式进行保护，则执行中断处理程序前需要执行 8 条 PUSH 指令，执行中断处理程序后还需要 8 条 POP 指令与之对应，程序将会变得很繁琐。

而本例只是执行一条“SETB RS0”指令，便将工作寄存器组由第 0 组变为第 1 组，R0～R7 的对应地址变为 08H～0FH，原来 R0～R7 对应的 00H～07H 的内容便被保护起来。

当执行完中断处理程序后，在恢复 PSW 值的同时，工作寄存器组又恢复为第 0 组，R0～R7 继续对应 00H～07H。

② “中断处理程序段”应根据中断任务的具体要求来编写。

③ 如果本中断服务程序不允许被其他的中断所中断，则将“中断处理程序段”前后的“SETB　EA”和“CLR　EA”两条指令去掉。

④ 若改变 Rn 的物理地址(修改 RS1 或 RS0)，则必须考虑堆栈区设置的问题。因为默认堆栈区和第 1 组工作寄存器组的物理地址(08H～0FH)重叠。

3. 中断返回

中断处理程序的最后一条执行指令必须是 RETI 。CPU 执行完这条指令时，一方面要清除中断响应时所置位的优先级激活触发器，另一方面要从堆栈中弹出断点地址放入 PC，使程序回到原断点处，继续执行原来的程序。

RET 指令虽然也能控制 PC 返回到原来中断的地方，但 RET 指令没有清 0 中断优先级激活触发器的功能。若采用 RET 指令作中断处理程序的最后一条执行指令，返回到主程序后中断控制系统会认为中断仍在进行，其后果是与此同级或低级的中断请求将不被响应。所以，**不能用 RET 指令代替 RETI 指令**。

另外，如果用户在中断服务程序中进行了入栈操作，则在 RETI 指令执行前应进行相应的出栈操作，即在**中断服务程序中 PUSH 指令与 POP 指令必须成对使用**，否则不能正确返回断点。

4. 中断请求的深入理解

MCS-51 单片机有 6 个中断请求标志位，即 IE0、IE1、TF0、TF1、RI、TI，对应着 5 个中断。当满足中断请求的条件时，由硬件置位(=1)，如：外中断引脚输入信号产生下降沿、计数器产生计数满额溢出等。如果这时满足中断响应条件，则会产生中断，我们可以编写中断服务程序完成相应的工作。

假设没有开放中断，在满足条件的情况下，中断请求标志位也会由单片机硬件置位。这样，我们就可以通过对中断请求标志位的查询，完成同样的任务，即编写查询程序来替代中断服务程序。只是在查询时需要注意对中断请求标志位的清 0。

表 5-3 给出了中断请求标志位的变化情况。

表 5-3 中断请求标志位的变化情况

中断源	中断请求标志位名称	置位条件	置位方法	清零方法	
				中断方式	查询方式
$\overline{\text{INT0}}$	IE0	单片机 12 引脚输入信号产生下降沿	硬件自动置位	硬件自动清 0	软件编程清 0
$\overline{\text{INT1}}$	IE1	单片机 13 引脚输入信号产生下降沿	硬件自动置位	硬件自动清 0	软件编程清 0
T0	TF0	计数器/定时器 0 产生计数溢出(计满)	硬件自动置位	硬件自动清 0	软件编程清 0
T1	TF1	计数器/定时器 1 产生计数溢出(计满)	硬件自动置位	硬件自动清 0	软件编程清 0
TxD	TI	发送缓冲器空(发完一个字节)	硬件自动置位	软件编程清 0	软件编程清 0
RxD	RI	接收缓冲器满(收到一个字节)	硬件自动置位	软件编程清 0	软件编程清 0

5.1.4 中断系统的应用

1. 中断程序的结构

MCS-51 单片机复位后，(PC)=0000H，即程序要从起始地址 0000H 开始执行，而 0003H、000BH、0013H、001BH、0023H 分别为各中断源的入口地址。为了跳过各中断源的入口地址，编程时应在 0000H 处使用一条无条件转移指令，跳转到主程序，主程序一般从地址 0030H 开始，参见图 4-1。也可以不设定主程序的开始地址，只设定标号，地址自然顺延，如下面的程序。

另外，各中断入口地址之间只差 8 个字节，如果中断服务程序的指令代码少于 8 个字节，则可以从规定的中断入口地址处直接编写中断服务程序。若中断服务程序的指令代码超过 8 个字节，则需要在中断入口地址处放置一条无条件转移指令，把中断服务程序跳转到合适的位置。

常用的中断程序结构如下：

```
        ORG     0000H
        LJMP    MAIN
        ORG     中断入口地址
        LJMP    INT
                ⋮
MAIN:           ⋮
        中断初始化程序
                ⋮
INT:    中断服务程序
                ⋮
```

2. 中断程序的内容

中断程序一般包含中断初始化程序和中断服务程序两部分。

对中断实现控制实质上就是对与中断有关的特殊功能寄存器进行管理和控制。单片机复位后，这些寄存器的内容都被清0，所有中断源都被屏蔽，要想开放CPU中断，允许某些中断源中断，必须对相关寄存器进行状态预置。中断初始化程序是指用户对相关寄存器中的各控制位进行赋值，即对如下内容进行初始化：

① 设置中断允许控制寄存器IE，允许相应中断源中断。

② 设置中断优先级寄存器IP，选择并分配所使用中断源的优先级。

③ 若是外部中断源，还要设置中断请求的触发方式IT0或IT1，以决定采用电平触发方式还是边沿触发方式。

中断服务程序要根据中断任务的具体要求进行编写。在编写中断服务程序时，要根据需要对一些重要寄存器或存储器的内容进行现场保护，在中断返回前还要进行现场恢复，并且中断服务程序的最后一条指令必须是RETI。

【例5-4】 试编写设置外部中断0为边沿触发的高优先级中断源的初始化程序。

参考程序1(采用位操作指令)：

```
SETB   EA     ；开启中断允许总控制位
SETB   EX0    ；允许外部中断0中断
SETB   PX0    ；设置外部中断0为高优先级
SETB   IT0    ；设置外部中断0为边沿触发方式
```

参考程序2(采用字节传送指令)：

```
MOV    IE，#81H       ；为IE赋值1000 0001B，即令EA和EX0为"1"
MOV    IP，#01H       ；为IP赋值0000 0001B，即令PX0为"1"
MOV    TCON，#01H     ；为TCON赋值0000 0001B，即令IT0为"1"
```

3. 外部中断源的应用程序

【例5-5】 如图5-6所示，89C51的P1口接有8个发光二极管，在外部中断1上接有一个按键，试编程实现如下功能：通过外部中断1，依次点亮8个发光二极管中的一个。

分析：因为8个发光二极管采用共阳极连接，所以若想通过外部中断1依次点亮8个发光二极管中的一个，首先要保证从P1口输出的数据中只有1位为"0"，其余7位为"1"，然后在中断服务程序中，将该数据循环左移或循环右移，即可实现依次点亮8个发光二极管中的一个。因为指令RL或RR的操作数必须是A，所以需要先将数据送给累加器A，然后通过A输出到P1口。

假设外部中断1采用边沿触发，高优先级。

参考程序：

```
        ORG    0000H
        LJMP   MAIN
        ORG    0013H        ；外中断1入口地址
        LJMP   INT          ；跳到中断服务程序
MAIN：  SETB   EA           ；开总中断
        SETB   EX1          ；开外中断1
```

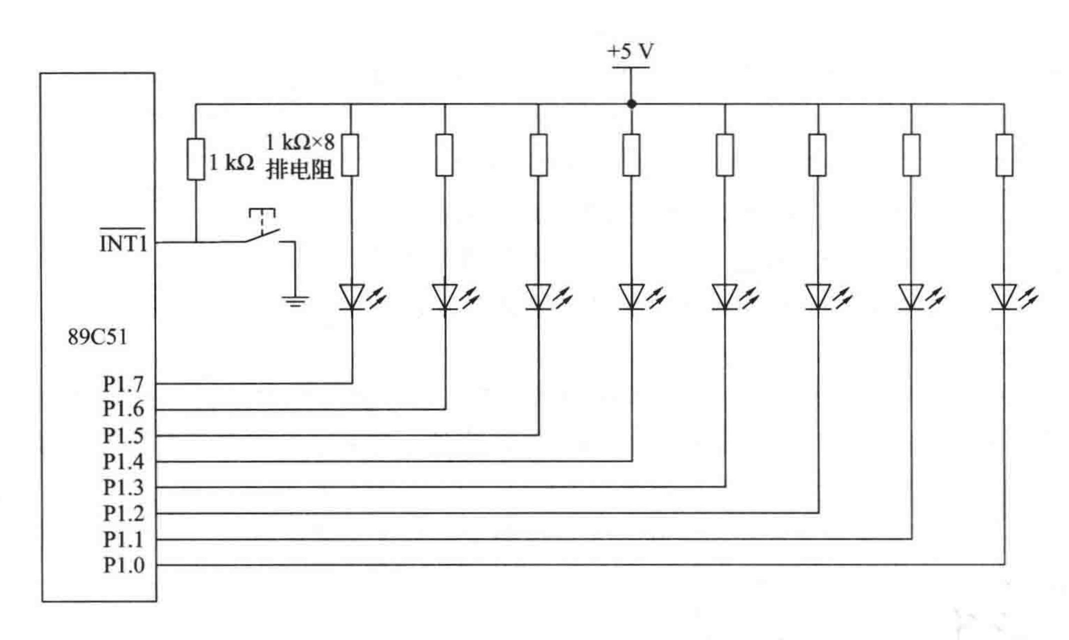


图 5-6 例 5-5 电路图

```
        SETB    PX1         ；外中断 1 设成高级别中断
        SETB    IT1         ；外中断 1 设成边沿触发
        MOV     A，#0FEH    ；给累加器 A 赋初值，预备右边第一个灯亮
HERE：  LJMP    HERE        ；原地等待中断申请
        ；必须加原地循环等待，否则不经实际中断而顺序执行中断服务程序，会造成程序崩溃
INT：   MOV     P1，A       ；输出到 P1 口
        RL      A           ；左环移一次
        RETI                ；中断返回，本程序是返回 HERE 语句
        END
```

特别需要注意的是：由于主程序没有实质性功能，所以我们在主程序的结尾处加了"HERE：LJMP HERE"(也可以用 LJMP \$)，用来等待中断，中断的断点也在这条语句上。如果没有这条语句，程序就会顺序执行中断服务程序，而不是从中断入口进入，这样就会产生程序运行错误。在实际的单片机应用系统中，"HERE：LJMP HERE"应该是一段具有实际功能的循环程序(参见例 5-6 和例 6-6)。

【例 5-6】 对于图 5-6 所示的电路，试编程实现如下功能：当无外部中断请求时，每隔 1s，依次点亮 8 个发光二极管中的 1 个；当有外部中断请求时，8 个发光二极管的显示状态改为闪烁显示(假设二极管点亮及熄灭的时间都是 1 s)，闪烁 5 次后，继续依次点亮。假设系统时钟频率为 12 MHz。

分析：根据题意，在无外部中断请求时，依次点亮 8 个发光二极管中的 1 个，所以数据循环左移或循环右移的程序段要放在主程序中。当有外部中断请求时，再根据要求修改 P1 口的状态。

本例参考 4.3.2 节的方法，编写一个延时子程序，以实现 1 s 的时间间隔。因为晶振频率为 $f_{osc}=12$ MHz，所以一个机器周期为 $12/f_{osc}=1\ \mu s$。执行一条 DJNZ 指令需要 2 μs，将该指令循环 250 次可以得到 500 μs。再将 500 μs 作为一个基本单位，循环 200 次可以得到

100 ms。再将 100 ms 作为一个基本单位，循环 10 次就得到 1 s，即 $10\times200\times250\times2\times1\ \mu s=1\ 000\ 000\ \mu s=1\ s$。

利用该方法计算得到的时间，忽略了赋值语句产生的时间延时，所以不够精确。5.2 节将介绍如何利用定时器/计数器获取更精确的定时时间。

参考程序：

```
         ORG    0000H
         LJMP   MAIN
         ORG    0013H          ；外中断 1 入口地址
         LJMP   INT
MAIN：   SETB   EA             ；开总中断
         SETB   EX1            ；开外中断 1
         SETB   IT1            ；外中断 1 设成边沿触发
         MOV    A，#0FEH       ；给累加器 A 赋初值，预备右边第一个灯亮
LOOP：   MOV    P1，A          ；输出到 P1 口
         LCALL  D_1s           ；延时 1 s
         RL     A              ；左环移一次
         LJMP   LOOP           ；循环显示
INT：    MOV    R7，#05H       ；置循环次数
L1：     MOV    P1，#00H       ；8 个发光二极管都点亮
         LCALL  D_1s           ；延时 1 s
         MOV    P1，#0FFH      ；8 个发光二极管都熄灭
         LCALL  D_1s           ；延时 1 s
         DJNZ   R7，L1         ；循环次数减"1"，不等于"0"则转到 L1
         RETI                  ；中断返回，返回的位置不固定
D_1s：   MOV    R6，#10        ；延时 1 s 子程序
D100ms： MOV    R5，#200
DL：     MOV    R4，#250
         DJNZ   R4，$
         DJNZ   R5，DL
         DJNZ   R6，D100ms
         RET                   ；子程序返回
         END
```

【例 5-7】 如图 5-7 所示，设计一个计数器，记录从 $\overline{\text{INT0}}$ 引脚输入的脉冲个数，存到一个 16 位的程序计数器中，当计数溢出时，将 P1.0 置位。

分析： $\overline{\text{INT0}}$ 输入脉冲会产生下降沿，正好和脉冲个数对应，所以我们可以设置下降沿触发方式，在中断服务程序里，将一个程序计数器+1，并判断是否溢出。在中断服务程序里只记一个数，退出等下一个中断到来。

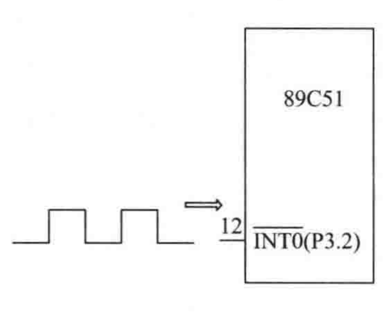


图 5-7　例 5-7 电路图

参考程序：

```
        ORG     0000H
        LJMP    MAIN
        ORG     0003H         ；外中断 0 入口
        LJMP    INT00
MAIN：  MOV     40H，#00H     ；程序计数器低 8 位，先清 0
        MOV     41H，#00H     ；程序计数器高 8 位，先清 0
        SETB    IT0           ；外中断 0 设成边沿触发
        SETB    EX0           ；开外中断 0
        SETB    EA            ；开总中断
M10：   LJMP    M10           ；等待中断
INT00： PUSH    PSW           ；堆栈保护
        MOV     A，40H        ；以下六句完成 16 位程序计数器+1 计算
        ADD     A，#01H
        MOV     40H，A
        MOV     A，41H
        ADDC    A，#00H
        MOV     41H，A
        JNC     INT10         ；没有进位，转移
        SETB    P1.0          ；有进位，P1.0 置 1
        CLR     EX0           ；关外中断 0，结束计数
INT10： POP     PSW           ；出栈
        RETI                  ；中断返回，本程序是返回 M10 语句
        END
```

4. 外部中断源的扩展及应用

MCS－51 单片机只有两个外部中断源的输入端，但实际应用中可能需要两个以上的外部中断源，这时就要对外部中断源的输入端进行扩展。扩展外部中断源的方法有定时器/计数器扩展法、中断和查询相结合的扩展法、硬件电路扩展法。这里仅介绍中断和查询相结合的扩展法。

利用 MCS－51 单片机的两个外部中断线，每个中断线可以通过“与”的关系连接多个外部中断源，同时利用 MCS－51 的 I/O 端口作为各中断源的识别标志，其原理如图 5－8 所示。

图 5－8 中的三个外部中断源在没有中断事件时输出高电平，通过与门输入到 $\overline{\text{INT0}}$ 端的电平为高电平，表示没有中断事件发生。当任意一个外部中断源有中断请求并送出一个低电平时，CPU 能立即响应该中断请求。在中断服务程序中，查询 P1.0、P1.1、P1.2 引脚上哪一个有低电平，判断是哪一个外部中断源发出的中断请求，并执行相应的中断处理程序（如查询到 P1.0 引脚上有低电平，则表明是外部中断源 X1 发出的中断请求，执行完 X1 的处理程序后返回主程序）。

【例 5－8】 用单片机监测 X1、X2、X3 三个外部设备在运行过程中是否有故障。无论哪个设备出现故障，都必须立刻处理，所以采用中断系统来检测这三个外部设备。当系统无故障时，3 个故障源输入端 X1～X3 全为低电平，对应的 3 个显示灯全灭；当某个设备出

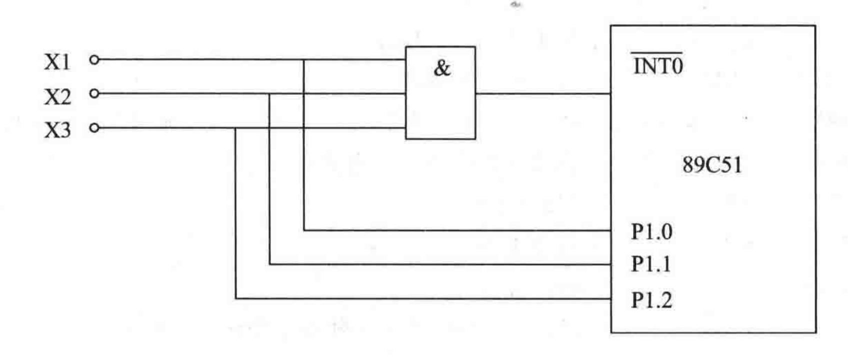


图 5－8　中断和查询相结合的外部中断源扩展电路

现故障时，其对应的输入端由低电平转为高电平，从而引起 MCS－51 单片机中断，中断服务程序的任务是判定故障，并点亮对应的发光二极管。实现上述功能的电路如图 5－9 所示。其中，发光二极管 LED1～LED3 对应 3 个输入端 X1～X3。3 个故障源通过或非门与 89C51 的外部中断 0 的输入端相连，同时，X1～X3 与 P1 口的 P1.0～P1.2 引脚相连，3 个发光二极管 LED1～LED3 分别与 P1 口的 P1.3～P1.5 相连。

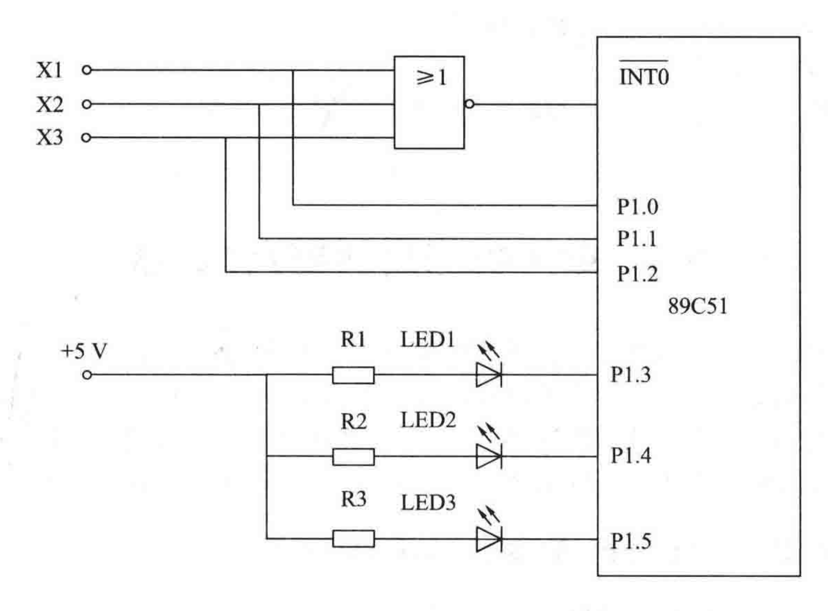


图 5－9　多故障检测电路

分析：根据题意，在进行故障检测之前，要先对 P1 口进行处理，不仅要使二极管不发光，还要保证外部中断 0 通道畅通，能够随时接收中断，为实现该功能可以将 P1 口与 11111000B 相或。进入中断后，要利用查询方式对中断源进行判断，处理方法为：若 P1.0 为高电平，说明 X1 有故障，则将 P1.3 位清 0，使二极管 LED1 导通发光，若 P1.0 为低电平，说明 X1 无故障(P1.0＝0)，继续判断 P1.1；若 X2 有故障(P1.1＝1)，则将 P1.4 位清 0，使二极管 LED2 导通发光，否则继续判断 P1.2；若 X3 有故障(P1.2＝1)，则将 P1.5 位清 0，使二极管 LED3 导通发光，否则，程序继续返回到主程序。

参考程序：

```
        ORG     0000H
        LJMP    MAIN
```

```
        ORG     0003H           ；外中断 0 入口地址
        LJMP    SERVE           ；转中断服务程序
MAIN：  ORL     P1，#0F8H       ；使 3 个发光二极管熄灭，同时保持 P1 低 3 位状态不变
        SETB    IT0             ；设置外部中断 0 为边沿触发方式
        SETB    EX0             ；开外中断 0
        SETB    EA              ；开总中断
        LJMP    $               ；循环等待
SERVE：PUSH    PSW             ；已出现故障，进入中断服务程序，保护 SFR
        PUSH    ACC
        JNB     P1.0，L1        ；判断设备 X1 是否有故障
        CLR     P1.3
L1：    JNB     P1.1，L2        ；判断设备 X2 是否有故障
        CLR     P1.4
L2：    JNB     P1.2，L3        ；判断设备 X3 是否有故障
        CLR     P1.5
L3：    POP     ACC             ；恢复 SFR
        POP     PSW
        RETI                    ；返回 $ 处的主程序继续等待检测
        END
```

5.2 单片机的定时器/计数器

在工业检测、控制领域中，许多场合都要用到计数或定时功能。例如对外部事件进行计数、产生精确的定时时间、作为串行口的波特率发生器等。MCS-51 单片机片内集成有两个可编程的定时器/计数器，即 T0 和 T1，它们都有定时和计数两种功能，可用于定时控制、延时、对外部事件计数和检测等场合。此外，T1 还可以作为串行口的波特率发生器。

5.2.1 定时器/计数器的基本结构和工作原理

1. 定时器/计数器的基本结构

定时器/计数器的基本结构如图 5-10 所示。

由图 5-10 可知，T0 由 2 个 8 位特殊功能寄存器 TH0 和 TL0 组成，T1 由 TH1 和 TL1 组成。T0 和 T1 的工作方式通过特殊功能寄存器 TMOD 来设定，T0 和 T1 的启动及停止由特殊功能寄存器 TCON 来控制。

2. 定时器/计数器的工作原理

图 5-11 所示是定时器/计数器的工作原理图。

由图 5-11 可知，定时器/计数器的核心部件是加 1 计数器。根据输入脉冲的来源不同，定时器/计数器分为两种工作模式：定时模式和计数模式。

当 T0 或 T1 设置为定时模式时，定时器对单片机内部的机器周期(系统内部振荡器输出脉冲的 12 分频)进行自动加 1 计数，将计数值乘以机器周期就得到了定时时间。

当 T0 或 T1 设置为计数模式时，计数器对来自输入引脚 T0(P3.4)或 T1(P3.5)的负脉

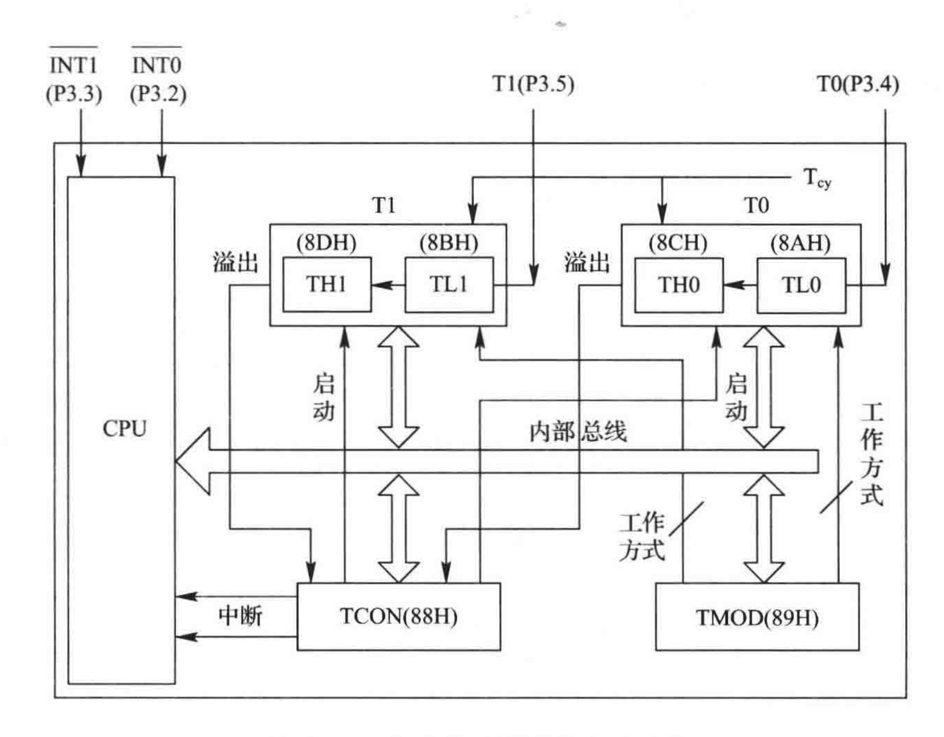


图 5-10　定时器/计数器的基本结构

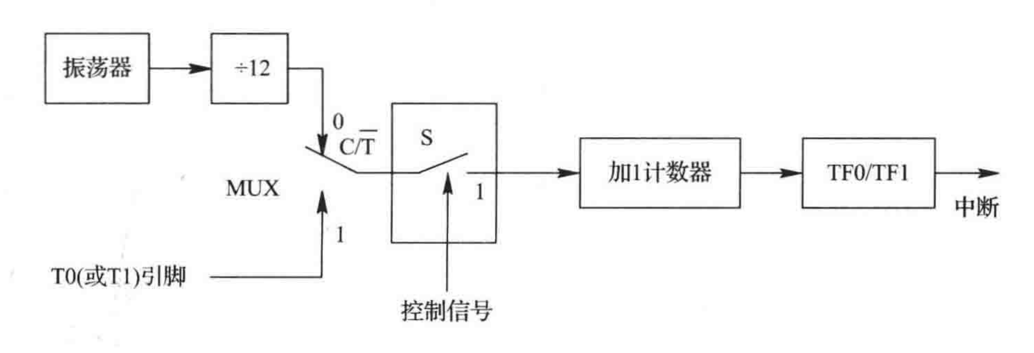


图 5-11　定时器/计数器的工作原理图

冲信号进行计数。为确保电平信号在变化之前至少被采样一次，要求输入的高电平和低电平信号至少应维持一个完整的机器周期，即它至少需要两个机器周期来识别一个高电平到低电平的跳变，所以计数模式时的计数频率最高为 f_{osc}(晶振)的 1/24。

定时器/计数器无论工作在定时模式还是计数模式，对输入脉冲的计数都不占用 CPU 的时间，除非定时器或计数器溢出，才能中断 CPU 的当前操作。定时器/计数器计满溢出时硬件令 TCON 中的 TF0 或 TF1 置 1，可以采用中断方式加以响应，也可以采用查询方式来处理。由此可见，定时器/计数器是单片机中效率高而且工作灵活的部件。

定时器本质上是一个计数器，由于初值可以根据需要设定，实际计数值就可控，则定时时间也是可控的。

5.2.2　定时器/计数器的控制与状态

MCS-51 单片机中有两个特殊功能寄存器与定时器/计数器有关，其中 TMOD 用于设置定时器/计数器的工作模式和工作方式，TCON 用于控制定时器/计数器的启动、停止

和中断请求。

1. 工作方式寄存器 TMOD

工作方式寄存器 TMOD 用于选择定时器/计数器的工作模式和工作方式，是不可位寻址的特殊功能寄存器，其字节地址为 89H。TMOD 的格式如下：

	D7	D6	D5	D4	D3	D2	D1	D0
(89H)								
TMOD	GATE	$C/\overline{T}$	M1	M0	GATE	$C/\overline{T}$	M1	M0
	← 定时器/计数器 T1 →				← 定时器/计数器 T0 →			

由此格式看出，TMOD 的 8 位分为两组，其中高 4 位用于控制 T1，低 4 位用于控制 T0。TMOD 中各位的含义如下：

① M1、M0：工作方式选择位。定时器/计数器有 4 种工作方式，由 M1、M0 进行设置，如表 5-4 所示。

表 5-4 定时器/计数器的 4 种工作方式

M1 M0	方 式	说 明
0 0	方式 0	13 位定时器/计数器
0 1	方式 1	16 位定时器/计数器
1 0	方式 2	自动重新装载的 8 位定时器/计数器
1 1	方式 3	T0 分成两个 8 位计数器，T1 停止计数

② $C/\overline{T}$：定时/计数模式选择位。

$C/\overline{T}=0$，设置为定时模式，对内部机器周期进行计数。

$C/\overline{T}=1$，设置为计数模式，对来自 T0、T1 引脚的外部脉冲信号进行计数。

③ GATE：门控位。

GATE=0 时，只要用软件使 TCON 中的运行控制位 TR0(或 TR1)为 1，就可以启动 T0(或 T1)。

GATE=1 时，既要用软件使运行控制位 TR0(或 TR1)为 1，还要使 $\overline{INT0}$或 $\overline{INT1}$引脚为高电平，才可以启动 T0(或 T1)。

MCS-51 单片机复位后，TMOD 所有位被清 0，使用前可以通过软件来设定它的工作方式。因为 TMOD 不能进行位寻址，所以只能采用字节操作指令设置 TMOD。

若要将 T0 设置为计数模式，按方式 2 工作，必须用如下指令对 TMOD 赋值：

```
MOV    TMOD, #06H
```

2. 中断请求标志寄存器 TCON

设定好 TMOD 后，定时器/计数器还不能进入工作状态，还必须通过设置 TCON 中的某些位来启动它。TCON 的低 4 位与外部中断设置有关，TCON 的高 4 位用于控制定时器/计数器的启动、停止和中断申请。TCON 格式如下：

	D7	D6	D5	D4	D3	D2	D1	D0
(88H)	8FH	8EH	8DH	8CH	8BH	8AH	89H	88H
TCON	TF1	TR1	TF0	TR0	IE1	IT1	IE0	IT0

TCON 中与定时器/计数器相关的高 4 位功能如下：

① TR0：T0 运行控制位。

当 TR0=1 时，启动 T0 计数；当 TR0=0 时，停止 T0 计数。

② TF0：T0 溢出中断请求标志位。

当启动定时器/计数器 T0 计数后，T0 从初值开始加 1 计数，当最高位产生溢出时，TF0 由硬件置“1”，向 CPU 申请中断。CPU 响应 TF0 中断时，TF0 由硬件清“0”；若不采用中断方式，则 TF0 必须由软件清“0”。

③ TR1：T1 运行控制位。其功能与 TR0 类似。

④ TF1：T1 溢出中断请求标志位。其功能与 TF0 类似。

5.2.3　定时器/计数器的工作方式

MCS-51 单片机的定时器/计数器共有 4 种工作方式(方式 0、1、2、3)，T0 和 T1 均可以设置在前 3 种工作方式(方式 0、1、2)，且 T0 和 T1 的工作原理相同，方式 0、1、2 对 T0 和 T1 均有效，只有 T0 才可以设置为工作方式 3。下面以 T0 为例介绍定时器/计数器的 4 种工作方式。

1. 方式 0

当 TMOD 的 M1M0 为 00 时，定时器/计数器工作于方式 0，如图 5-12 所示。

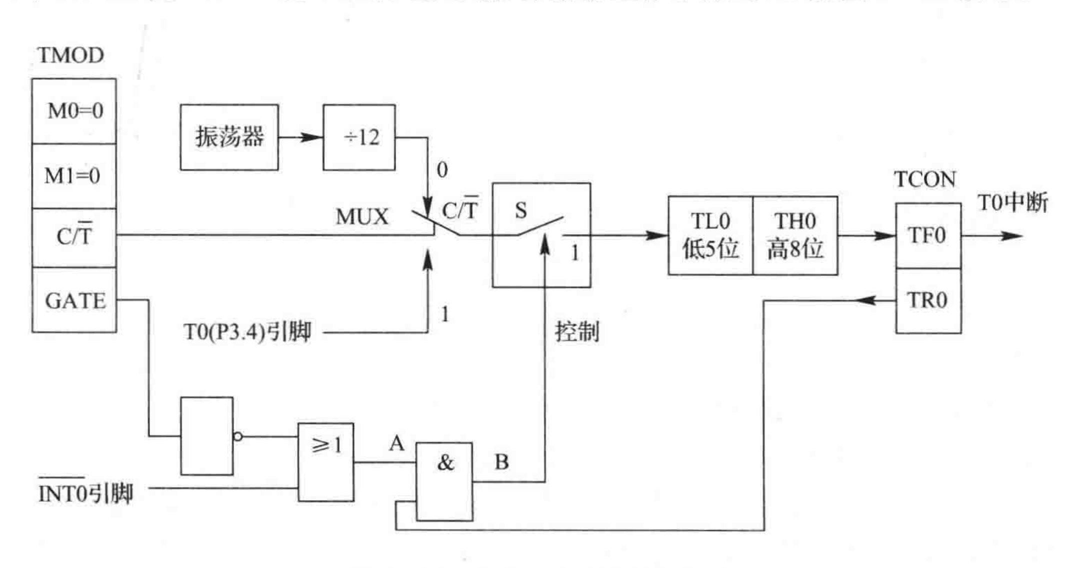


图 5-12　方式 0 的逻辑结构框图

方式 0 为 13 位计数器，由 TL0 的低 5 位和 TH0 的高 8 位组成。当 TL0 的低 5 位计满溢出时向 TH0 进位，当 TH0 计满溢出时，将中断标志位 TF0 置位，并向 CPU 申请中断。该方式是为兼容 MCS-48 而设的，在实际应用中几乎不再使用。

方式 0 和方式 1 的结构与操作基本相同，其差别仅仅在于计数的位数不同。方式 0 为

13 位计数器，而方式 1 为 16 位计数器，详细内容参考方式 1。

2. 方式 1

当 TMOD 的 M1M0 为 01 时，定时器/计数器工作于方式 1，如图 5－13 所示。

方式 1 为 16 位计数器。T0 工作在方式 1 时，由 TL0 的低 8 位和 TH0 的高 8 位组成。

当 TL0 的低 8 位计满溢出时向 TH0 进位，当 TH0 计满溢出时，将中断标志位 TF0 置位，并向 CPU 申请中断。

当 $C/\overline{T}$=0 时，多路转换开关 MUX 接振荡器 12 分频的输出端，即 T0 工作于定时模式，对机器周期进行计数。计数值乘以机器周期等于定时时间。

当 $C/\overline{T}$=1 时，多路转换开关 MUX 接 P3.4 引脚，接收外部输入的信号，即 T0 工作在计数模式，对外部脉冲进行计数。

当 GATE=0 时，$\overline{INT0}$被屏蔽，对或门输出不产生影响，此时仅由 TR0 来控制 T0 的开启和关闭。当 TR0=1 时，T0 将从初值开始加 1 计数。当 TR0=0 时，停止计数。

当 GATE=1 时，T0 的开启与关闭取决于 $\overline{INT0}$和 TR0 相与的结果，即只有当 TR0=1 且 $\overline{INT0}$=1 时，T0 才开始工作，否则停止计数。利用该特点可以测量在 $\overline{INT0}$端出现的正脉冲的宽度。

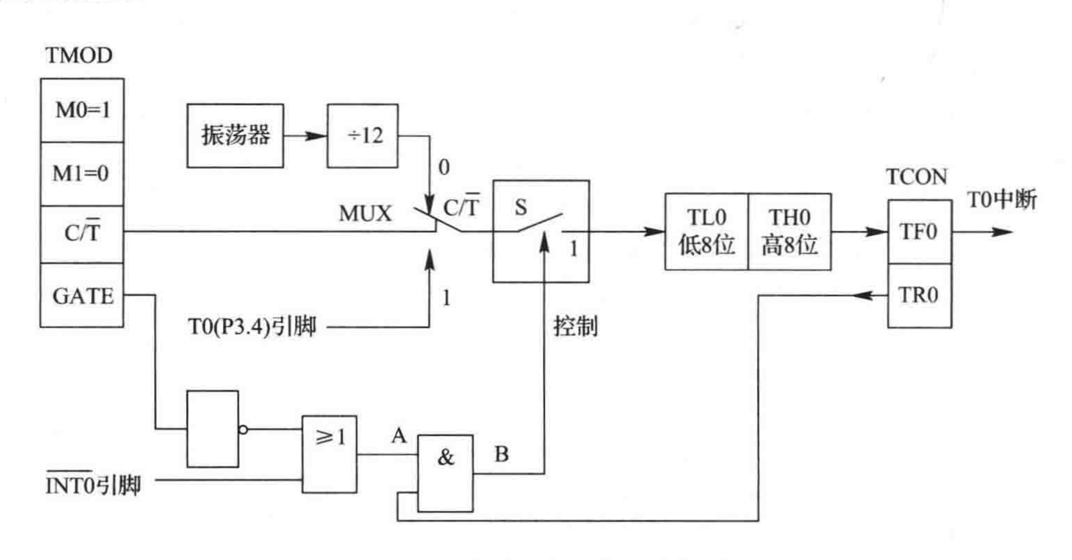


图 5－13　方式 1 的逻辑结构框图

3. 方式 2

当 TMOD 的 M1M0 为 10 时，定时器/计数器工作于方式 2，如图 5－14 所示。

方式 2 是具有自动重装初值功能的 8 位计数器。

方式 0 和方式 1 计数溢出后，TH0 和 TL0 的初值均变为 0，所以在编制循环程序时需要反复设定初值，既不方便又影响定时精度。方式 2 将 TL0 作为 8 位计数器，TH0 仅保存计数初值而不参与计数。初始化时 TL0 和 TH0 的值均是初值，且相等。当 TL0 计满溢出时，CPU 在将溢出中断请求标志位 TF0 置“1”的同时自动将 TH0 中的初值重新装入 TL0 中，使 TL0 从初值重新开始计数。

方式 2 省去了重装初值的时间，可以实现精确的定时。其中 T1 采用方式 2，常用于产生单片机串行通信中的波特率，并且只有 T1 才能产生波特率信号。

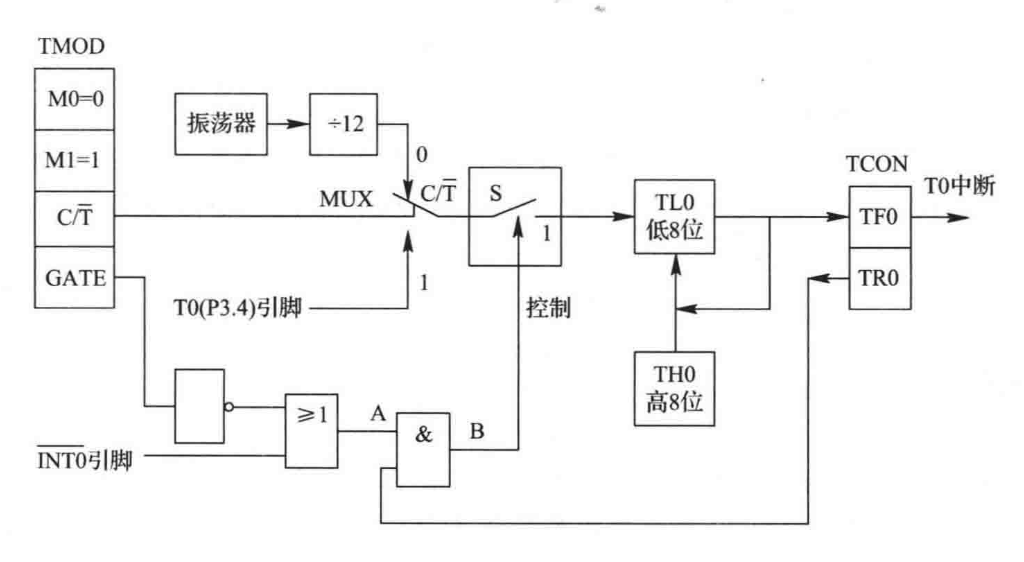


图 5-14 方式 2 的逻辑结构框图

4. 方式 3

当 TMOD 的 M1M0 为 11 时，定时器/计数器工作于方式 3，如图 5-15 所示。

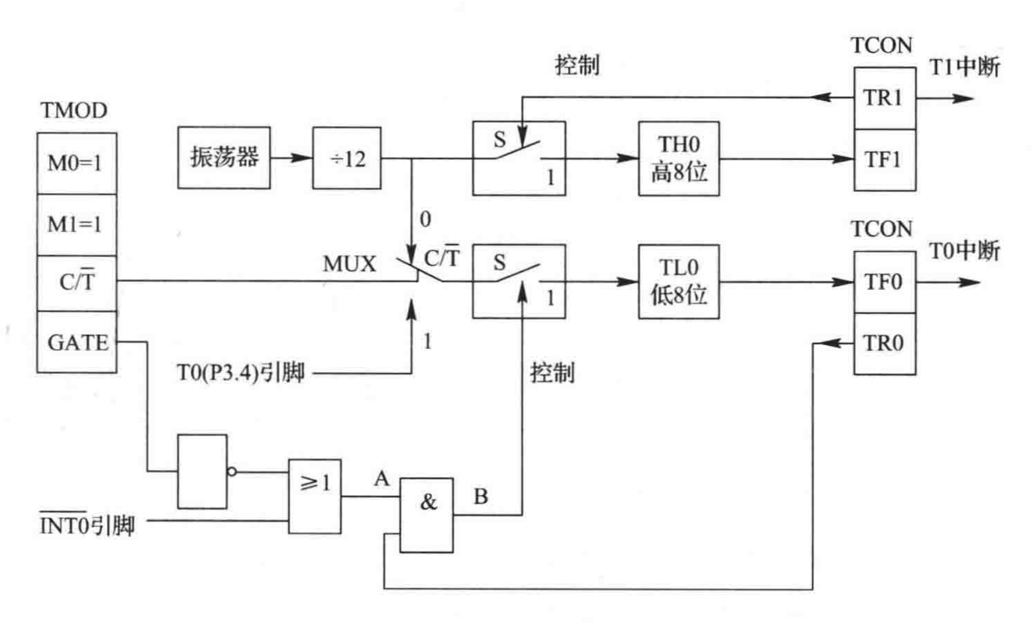


图 5-15 方式 3 的逻辑结构框图

方式 3 只适用于 T0，当 T0 工作在方式 3 时，TL0 和 TH0 成为两个独立的 8 位计数器。TL0 使用 T0 的所有控制位：C/$\overline{\text{T}}$、GATE、TR0、TF0 和 $\overline{\text{INT0}}$，它的操作与方式 0 和方式 1 类似。TH0 要借用 T1 的运行控制位 TR1 和溢出标志位 TF1，且只能作 8 位定时器，即只能对机器周期进行计数，而不能对外部脉冲进行计数。

当 T0 工作在方式 3 时，T1 可以工作在方式 0、1、2。因为它不能置位 TF1，所以只能用于不需要中断控制的场合，或用作串行口的波特率发生器。通常，当 T1 用作串行口波特率发生器时，T0 才定义为方式 3，以增加一个 8 位计数器。

方式 1 和方式 2 是实际应用中最常用的两种方式，应重点掌握。

5.2.4 定时器/计数器的初值计算和初始化

1. 计数初值的计算

MCS-51单片机的定时器/计数器具有4种工作方式，因为不同工作方式下计数器的位数不同，所以计数的最大值也不同。假设当前工作方式下的最大计数值用M表示，则各种工作方式的最大计数值如下：

方式0： $M=2^{13}=8192$

方式1： $M=2^{16}=65\ 536$

方式2： $M=2^{8}=256$

方式3： $M=2^{8}=256$

MCS-51单片机采用特殊功能寄存器TH0和TL0来存放定时器T0的计数初值，TH1和TL1来存放定时器T1的计数初值。T0和T1从初值开始以"加1"方式进行计数，当计数值达到最大值时会产生溢出，并产生中断申请。由于计数初值可以由软件设定，所以定时时间是可控的。假设用X表示计数初值，N表示能产生溢出的计数值，可得计数模式下，计数值N与计数初值X之间的关系式：

$$X=M-N \tag{5-1}$$

对于定时模式，计数值N乘以机器周期 T_{cy}（$T_{cy}=12/f_{osc}$）等于定时时间t，即

$$t=N\times T_{cy}=(M-X)\times T_{cy} \tag{5-2}$$

由此式可以得到计数初值X的表达式为

$$X=M-t\times f_{osc}/12 \tag{5-3}$$

其中：f_{osc}为晶振频率；M为最大计数值；t为所需的定时时间。

2. 定时器/计数器的初始化

由于定时器/计数器是可编程的，因此在利用定时器/计数器进行定时或计数之前，要先通过软件对其进行初始化。初始化的步骤如下：

(1) 根据要求为工作方式寄存器TMOD赋值，以设定T0和T1的工作模式和工作方式。

(2) 根据需要计算计数初值，并将初值送入TH0、TL0、TH1、TL1。

(3) 根据需要给IE和IP赋值，以开放相应中断和设置中断优先级。

(4) 设置TCON中的TR0、TR1，以启动或禁止T0、T1的运行。

5.2.5 定时器/计数器的应用

1. 初值计算和初始化

【例5-9】 假设定时时间为5 ms，单片机的主频(晶振频率)为6 MHz，使用T0，求方式1的计数初值。

解：

$$T_{cy}=12/(6\ \text{MHz})=2\ \mu\text{s}$$

$$X=M-\frac{t}{T_{cy}}=M-\frac{5\ \text{ms}}{2\ \mu\text{s}}=M-2500$$

对于方式1，

$$X=2^{16}-2500=63\ 036=0\text{F63CH}$$

其中低 8 位 3CH 送入 TL0，高 8 位 F6H 送入 TH0。

【例 5-10】 假设 T0 为定时模式，按方式 2 工作，TH0、TL0 的初值均为 0FH，且允许 T0 中断，试对该定时器进行初始化。

参考程序：

```
MOV     TMOD，#02H      ；置 T0 为定时器方式 2
MOV     TL0，#0FH       ；置计数初值
MOV     TH0，#0FH
SETB    EA              ；CPU 开中断
SETB    ET0             ；允许 T0 中断
SETB    TR0             ；启动 T0 工作
```

2. 计数应用

【例 5-11】 在某工厂的一条自动饮料生产线上，每生产 12 瓶饮料，就需要发出一个包装控制信号自动执行装箱操作，试编写程序完成这一计数任务。假设用 T0 完成计数，用 P1.0 发出控制信号，包装流水线示意图如图 5-16 所示。

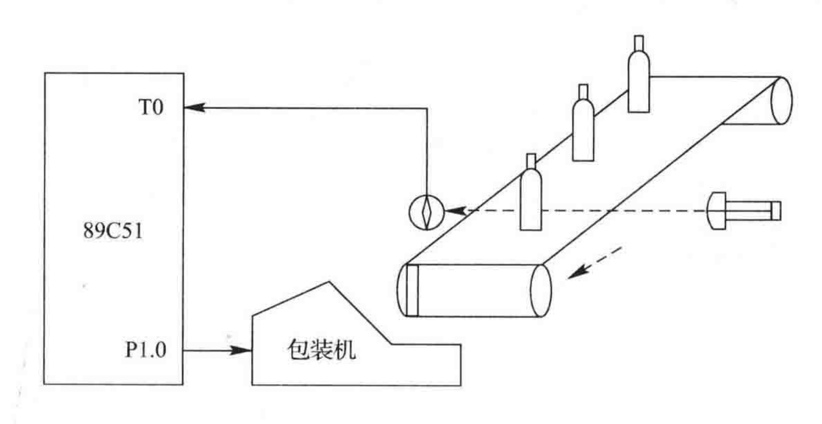


图 5-16　包装流水线示意图

分析：(1) 选择工作方式：因为计数值为 12，所以选用 T0 的工作方式 2 来完成此任务。假设此时 T1 不工作，则方式控制字为 TMOD=06H。

(2) 求计数初值 X：

$$X=256-12=244=F4H$$

因此，TL0 和 TH0 的初值都为 F4H。

本题可以采用查询和中断两种方式进行编程，如图 5-17 所示。无论采用哪种方式，都要先按照定时器/计数器的初始化步骤对相关特殊功能寄存器进行设置，并且当计数值达到要求后，对 P1.0 进行设置。本例假设 P1.0 为高电平时发出控制信号，且该控制信号只要持续 2 个机器周期以上即可。

需要注意的是：

① 若采用中断方式进行控制，一旦计数值达到要求，计数器便产生溢出，从而进入中断服务程序，同时硬件自动将中断标志位 TF0 清 0。

② 若采用查询方式进行控制，程序需要不断对中断标志位 TF0 进行查询：若 TF0 为 0，计数器未溢出，则程序在当前指令循环；若 TF0 为 1，计数器溢出，则执行下面的程序，

此时 T0 虽然发出中断请求，但因为 IE 中的相应开关断开，所以该中断请求不会得到响应，为此需要利用软件将 TF0 清 0。

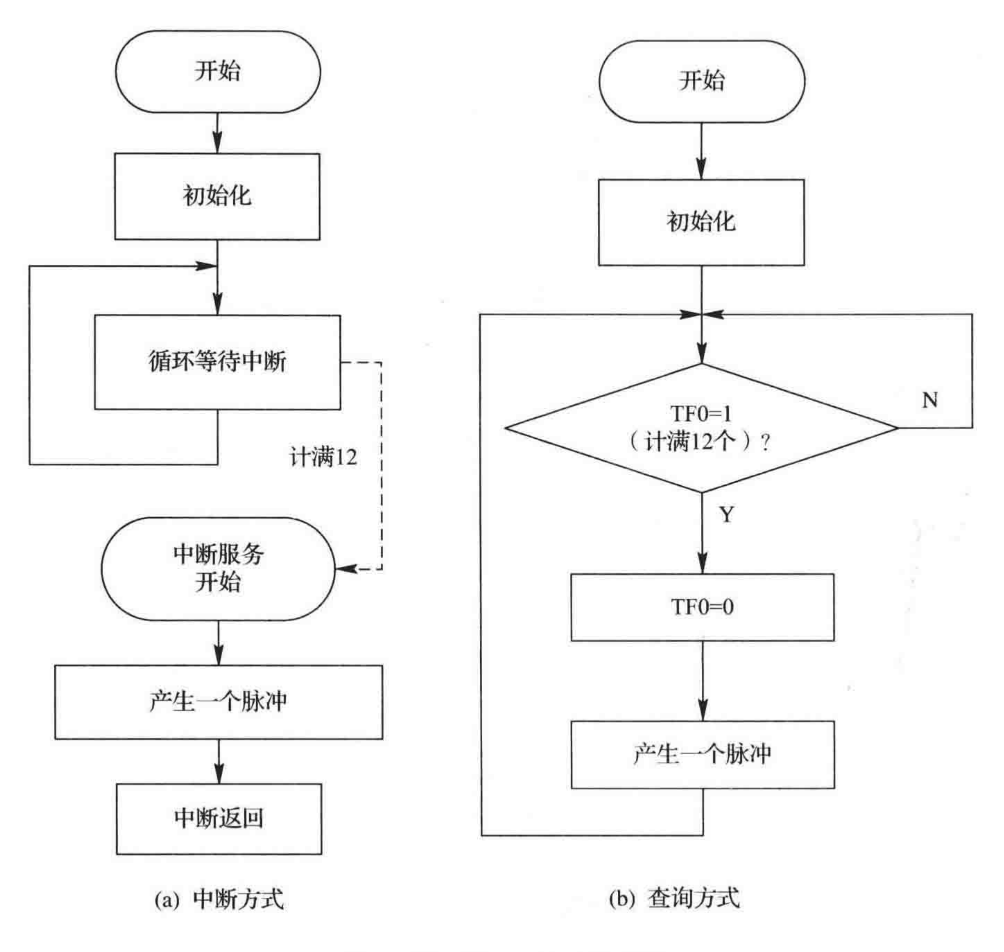


图 5－17　例 5－11 的流程图

参考程序 1(采用中断方式)：

```
        ORG    0000H
        LJMP   MAIN
        ORG    000BH          ；T0 入口地址
        LJMP   DVT0           ；转中断服务程序
MAIN：  MOV    TMOD，#06H     ；设置 T0 工作在方式 2，计数模式
        MOV    TH0，#0F4H     ；装入计数初值
        MOV    TL0，#0F4H
        SETB   ET0            ；T0 开中断
        SETB   EA             ；CPU 开中断
        SETB   TR0            ；启动 T0
        LJMP   $              ；等待中断
DVT0：  SETB   P1.0           ；产生脉冲信号高电平
        NOP                   ；以下两句用于延迟 2 个机器周期，产生可靠的控制脉冲
        NOP
```

```
        CLR     P1.0            ；产生脉冲信号低电平
        RETI                    ；中断返回，重新开始计数
        END
```

参考程序 2(采用查询方式)：

```
        ORG     0000H
        MOV     TMOD，#06H      ；设置T0工作在方式2，计数模式
        MOV     TL0，#0F4H      ；装入计数初值
        MOV     TH0，#0F4H
        MOV     IE，#00H        ；关闭中断允许寄存器
        SETB    TR0             ；启动T0
LOOP：  JNB     TF0，LOOP       ；判断T0是否计数溢出
        CLR     TF0             ；清T0中断申请
        SETB    P1.0            ；产生脉冲信号高电平
        NOP                     ；以下两句用于延迟2个机器周期，产生可靠的控制脉冲
        NOP
        CLR     P1.0            ；产生脉冲信号低电平
        LJMP    LOOP            ；循环返回
        END
```

【例 5-12】 利用定时器 T1 的方式 2 对外部信号计数，要求每计满 100 个数将 P1.7 取反。

分析：(1) 确定方式字：假设此时 T0 不工作，则 T1 工作在方式 2 的控制字为 TMOD=60H。

(2) 计算初值：

$$X=2^8-100=156=9CH$$

因此，TL1 和 TH1 的初值都为 9CH。

本例与例 5-11 类似，都是实现计数功能，所以实现方法类似，既可以采用中断方式又可以采用查询方式，这里只介绍中断方式，查询方式的编程方法可以参考例 5-11。

参考程序：

```
        ORG     0000H
        LJMP    MAIN
        ORG     001BH           ；T1中断服务程序入口
        CPL     P1.7            ；P1.7位取反
        RETI                    ；中断返回
                                ；因为该中断服务程序很短，小于8个字节，
                                ；故可以直接在中断向量区编程，没跳转
MAIN：  MOV     TMOD，#60H      ；主程序开始，设置T1工作在方式2，计数模式
        MOV     TL1，#9CH       ；T1赋初值
        MOV     TH1，#9CH
        SETB    ET1             ；允许T1中断
        SETB    EA              ；CPU开中断
        SETB    TR1             ；启动T1
```

```
HERE：LJMP    HERE
      END
```

3. 定时应用

【例 5－13】 假设系统时钟频率为 6 MHz，现欲利用定时器 T0 每隔 1 ms 产生宽度为 1 个机器周期的正脉冲，并由 P1.0 送出，如图 5－18 所示，请编程实现该功能。

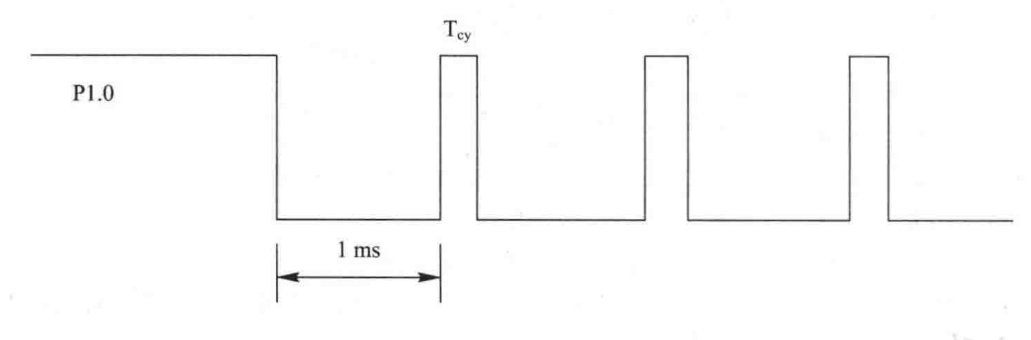


图 5－18　例 5－12 输出波形图

分析：(1) 选择工作方式：因为 $T_{cy}=12/f_{osc}=2\ \mu s$，由定时器各种工作方式的特性，可计算出方式 0 最长可定时 16.384 ms，方式 1 最长可定时 131.072 ms，方式 2、3 最长可定时 512 μs。

本例中定时时间 t＝1 ms，故选择 T0 的工作方式 1 来完成此任务。假设此时 T1 不工作，则方式控制字为 TMOD＝01H。

(2) 计算初值 X：

$$X=2^{16}-1000\ \mu s/2\ \mu s = 65\ 536-500 = 65\ 036 = \text{FE0CH}$$

因此 T0 的初值为 TH0＝0FEH，TL0＝0CH。

本例与例 5－11 类似，也可以采用查询和中断两种方式进行编程，处理过程也大致相同，与例 5－11 最大的不同之处在于本例的定时器/计数器将工作于定时模式。

参考程序 1(采用查询方式)：

```
        ORG     0000H
        CLR     P1.0            ；将输出口 P1 的第 0 位清 0(输出脉冲的起始值)
        MOV     TMOD，#01H      ；设置 T0 工作在方式 1，定时模式
        MOV     TH0，#0FEH      ；T0 赋初值
        MOV     TL0，#0CH
        SETB    TR0             ；启动 T0
LOOP：  JNB     TF0，LOOP       ；判断 T0 是否计数溢出
        CLR     TF0             ；将 TF0 清 0
        SETB    P1.0            ；将 P1.0 位置 1，输出正脉冲
        CLR     P1.0            ；将 P1.0 位清 0，因为上一条指令是单周期指令，所以
                                ；正脉冲宽度为 1 个机器周期
        MOV     TH0，#0FEH      ；重新装载 TH0 的初值
        MOV     TL0，#0CH       ；重新装载 TL0 的初值
        LJMP    LOOP
        END
```

参考程序 2(采用中断方式)：

```
        ORG     0000H
        LJMP    MAIN
        ORG     000BH              ；T0 的入口地址
        LJMP    T0INT
MAIN：  CLR     P1.0               ；将 P1 口的第 0 位清 0(输出脉冲的起始值)
        MOV     TMOD，#01H         ；设置 T0 工作在方式 1，定时模式
        MOV     TH0，#0FEH         ；T0 赋初值
        MOV     TL0，#0CH
        MOV     IE，#82H           ；ET0=1，EA=1，可以用 SETB EA 和 SETB ET0 代替
        SETB    TR0                ；启动定时器 0
LOOP：  LJMP    LOOP               ；等待中断
T0INT： SETB    P1.0               ；中断服务程序，P1.0 输出高电平，占 1 个机器周期
        CLR     P1.0               ；P1.0 输出低电平
        MOV     TH0，#0FEH         ；重新装载 TH0 的初值
        MOV     TL0，#0CH          ；重新装载 TL0 的初值
        RETI                       ；中断返回
        END
```

【例 5－14】 设时钟频率为 12 MHz，编程实现用定时器 T0 产生 50 Hz 的方波，并由 P1.0 输出此方波。

分析：(1) 选择工作方式：因为 $T_{cy}=12/f_{osc}=1\ \mu s$，由定时器各种工作方式的特性，可计算出方式 0 最长可定时 8.912 ms，方式 1 最长可定时 65.536 ms，方式 2、3 最长可定时 256 μs。

因为 50 Hz 方波的周期为 20 ms，所以只要每隔 10 ms 变化一次 P1.0 的电平，就可获得 50Hz 的方波，即本例的定时时间为 10 ms，所以选择 T0 的工作方式 1 来完成此任务。假设此时 T1 不工作，则方式控制字为 TMOD=01H。

(2) 计算初值 X：

$$X=2^{16}-\frac{10\ 000\ \mu s}{1\ \mu s}=65\ 536-10\ 000=55\ 536=D8F0H$$

因此 T0 的初值为 TH0=0D8H，TL0=0F0H。

参考程序：

```
        ORG     0000H
        LJMP    MAIN
        ORG     000BH              ；T0 入口地址
        LJMP    T0INT
MAIN：  MOV     TMOD，#01H         ；设置 T0 工作在方式 1，定时模式
        MOV     TH0，#0D8H         ；赋初值
        MOV     TL0，#0F0H
        MOV     IE，#82H           ；EA=1，ET0=1
        SETB    TR0                ；T0 启动
```

```
LOOP: LJMP    LOOP          ；等待中断
T0INT:CPL     P1.0          ；中断服务程序，第一条语句将 P1.0 位的内容反相
      MOV     TH0，#0D8H    ；重新装载 TH0 的初值
      MOV     TL0，#0F0H    ；重新装载 TL0 的初值
      RETI                  ；中断返回
      END
```

【例 5-15】 假设系统时钟频率为 $f_{osc}=3$ MHz，利用定时器编程实现如下功能：从 P1.7引脚输出一个脉冲波形，高电平持续 50 ms，低电平持续 150 ms，如图 5-19 所示。

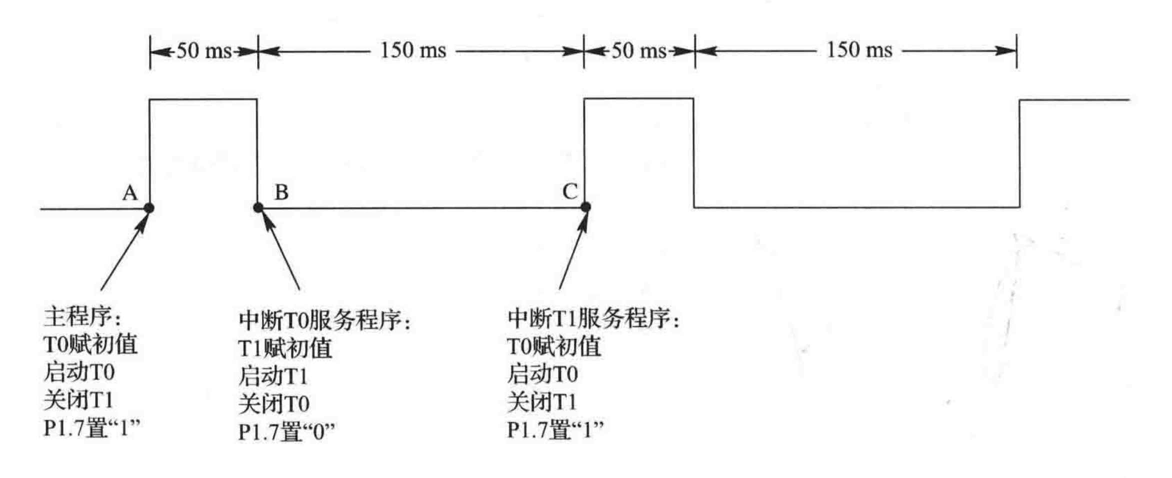


图 5-19　例 5-15 的波形图

分析 1： 因为高电平和低电平持续时间不同，考虑用两个定时器/计数器 T0 和 T1。因为 $T_{cy}=12/f_{osc}=4\ \mu s$，考虑到定时时间较长，只能选择方式 1，最大计数值是 65 536，最大定时时间是 65 536×4 μs=262 ms。两个时间均可以用一次中断来定时。计数初值分别是

$$X_1=2^{16}-50\ ms/4\ \mu s=65\ 536-12\ 500=53\ 036=CF2CH$$

$$X_2=2^{16}-150\ ms/4\ \mu s=65\ 536-37\ 500=28\ 036=6D84H$$

T0 定时 50 ms，则 TH0=CFH，TL0=2CH；T1 定时 150 ms，则 TH1=6DH，TL1=84H。

① 在初始化程序中，对 T0 赋初值，启动 T0，关闭 T1，P1.7 置"1"，T0 开始第一个 50 ms定时。

② 当定时 50 ms 到时，进入中断 T0INT，对 T1 赋初值，启动 T1，关闭 T0，P1.7 置"0"，T1 开始第一个 150 ms 定时。

③ 当定时 150 ms 到时，进入中断 T1INT，对 T0 赋初值，启动 T0，关闭 T1，P1.7 置"1"，T0 开始第二个 50 ms 定时。

……

参考程序 1：

```
        ORG     0000H
        LJMP    MAIN
        ORG     000BH           ；T0 入口地址
        LJMP    T0INT
        ORG     001BH           ；T1 入口地址
```

```
        LJMP    T1INT
        ORG     0030H           ；主程序开始地址
MAIN：  MOV     TMOD，#11H      ；设置 T0 和 T1 工作在方式 1
        SETB    P1.7            ；P1.7 产生高电平
        MOV     TH0，#0CFH      ；T0 赋初值
        MOV     TL0，#2CH
        SETB    TR0             ；启动 T0
        SETB    ET0             ；开 T0 中断
        SETB    ET1             ；开 T1 中断
        SETB    EA              ；开总中断
HERE：  LJMP    HERE            ；等待中断
T0INT： NOP                     ；T0 中断服务程序
        CLR     P1.7            ；P1.7 产生低电平
        CLR     TR0             ；关闭 T0
        MOV     TH1，#6DH       ；T1 赋初值
        MOV     TL1，#84H
        SETB    TR1             ；启动 T1
        RETI                    ；T0 中断返回
T1INT： NOP                     ；T1 中断服务程序
        SETB    P1.7            ；P1.7 产生高电平
        CLR     TR1             ；关闭 T1
        MOV     TH0，#0CFH      ；T0 赋初值
        MOV     TL0，#2CH
        SETB    TR0             ；启动 T0
        RETI                    ；T1 中断返回
        END
```

分析 2：因为 150 ms 是 50 ms 的 3 倍，可以用一个定时器/计数器 T0。选择方式 1，最大计数值是 65 536，最大定时时间是 65 536×4 μs=262 ms。定时时间 50 ms。计数初值是

$$X=2^{16}-50\ \text{ms}/4\ \mu\text{s}=65\ 536-12\ 500=53\ 036=\text{CF2CH}$$

则 TH0=CFH，TL0=2CH。

设定一个程序计数器 R7，每次中断到来时+1，然后判断是否"=1"或"=4"，分别对应 50ms 和 150ms，"=4"完成后给 R7 清 0。

……

参考程序 2：

```
        ORG     0000H
        LJMP    MAIN
        ORG     000BH           ；T0 入口地址
        LJMP    T0INT
        ORG     0030H           ；主程序开始地址
MAIN：  MOV     TMOD，#01H      ；设置 T0 工作在方式 1
        SETB    P1.7            ；P1.7 产生高电平
```

```
        MOV     TH0, #0CFH          ; T0 赋初值
        MOV     TL0, #2CH
        SETB    TR0                 ; 启动 T0
        SETB    ET0                 ; 开 T0 中断
        SETB    EA                  ; 开总中断
        MOV     R7, #00H            ; 程序计数器清零
HERE:   LJMP    HERE                ; 等待中断
T0INT:  NOP                         ; T0 中断服务程序
        INC     R7                  ; 程序计数器 R7 加 1
        CJNE    R7, #01H, T10       ; 判断 50 ms 是否到时
        CLR     P1.7                ; 输出低电平
        LJMP    T20
T10:    CJNE    R7, #04H, T20       ; 判断 150 ms 是否到时
        SETB    P1.7                ; 输出高电平
        MOV     R7, #00H            ; 程序计数器 R7 清零
T20:    MOV     TH0, #0CFH          ; T0 赋初值
        MOV     TL0, #2CH
        RETI                        ; T0 中断返回
        END
```

【例 5-16】 89C51 的 P2 口接了 8 个发光二极管，如图 5-20 所示。要求通过定时器 1 实现 8 个发光二极管每隔 1 s 从右向左依次循环点亮。假设系统时钟频率为 12 MHz。

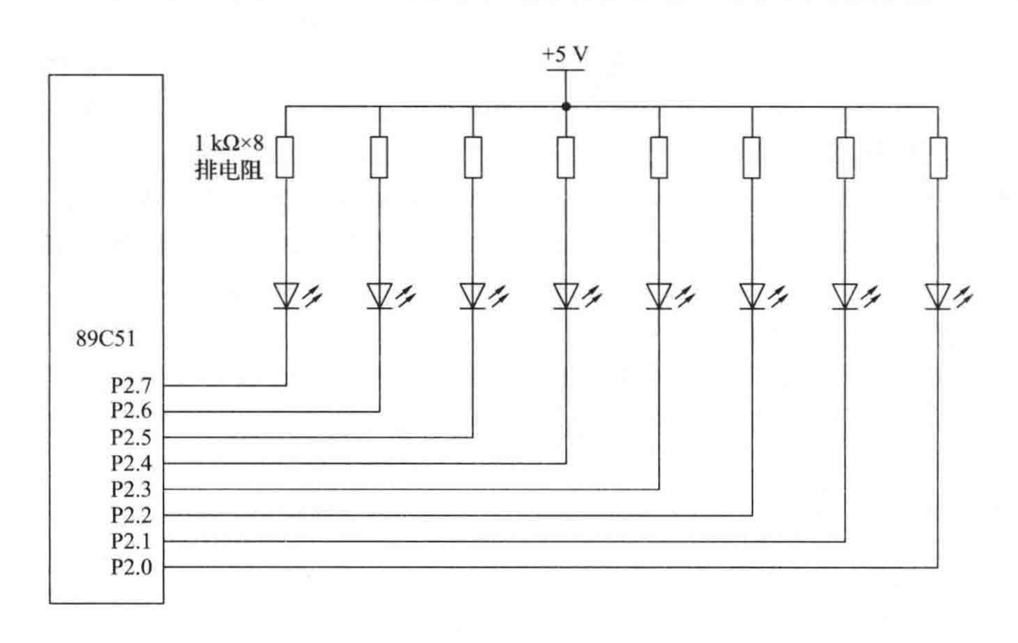


图 5-20 例 5-16 电路图

分析：(1) 选择工作方式：本例定时时间较长，超出方式 0～3 的最长定时范围，所以不能直接采用方式 0～3 实现该定时功能。我们可以将这 1s 定时时间分成若干份，每份定时时间在所选工作方式的定时范围内，然后用软件进行计数来实现该功能。

本例选择 T1 的工作方式 1，每隔 50 ms 中断一次，中断 20 次为 1 s。假设此时 T0 不

工作，则方式控制字为 TMOD=10H。

(2) 计算计数初值：

$$X=2^{16}-\frac{50\ 000}{1}=15\ 536=3CB0H$$

因此 TH1=3CH，TL1=B0H。

(3) 20 次计数的实现：采用循环程序的方法实现中断 20 次计数。

参考程序：

```
        ORG     0000H
        LJMP    MAIN
        ORG     001BH           ；T1 入口地址
        LJMP    T1INT
MAIN：  MOV     TMOD，#10H      ；设置 T1 工作于方式 1
        MOV     TH1，#3CH       ；T1 赋初值
        MOV     TL1，#0B0H
        SETB    EA              ；开总中断
        SETB    ET1             ；开 T1 中断
        SETB    TR1             ；启动 T1
        MOV     R7，#20         ；循环计数器初值 20，50 ms×20=1s
        MOV     A，#0FEH        ；给累加器 A 赋初值，预备右边第一个灯亮
        MOV     P2，A           ；输出到 P2 口
        LJMP    $               ；等待中断
TIMER1：DJNZ    R7，RETURN      ；判断是否到达循环次数
        RL      A               ；左环移一次
        MOV     P2，A           ；输出到 P2 口
        MOV     R7，#20         ；置循环次数
RETURN：MOV     TH1，#3CH       ；重新装载 TH1 的初值
        MOV     TL1，#0B0H      ；重新装载 TL1 的初值
        RETI                    ；中断返回
        END
```

3. 门控位的应用

【例 5-17】 利用 T0 测量 $\overline{INT0}$引脚出现的正脉冲的宽度，并将测量结果(以机器周期的形式)存放在 30H 和 31H 两个单元中。

分析： 要想测量 $\overline{INT0}$引脚出现的正脉冲的宽度，首先要将 T0 设置为方式 1 的定时模式，TR0 置 1，门控位 GATE 置 1，T0 初值取 0。这样当 $\overline{INT0}$引脚变为高电平时采用外触发方式启动 T0 定时，即对机器周期计数；当外部 $\overline{INT0}$引脚变为低电平时停止 T0 计数，这时 TH0 和 TL0 中的值就是 $\overline{INT0}$引脚为高电平期间所经过的机器周期数。处理过程如图 5-21 所示。

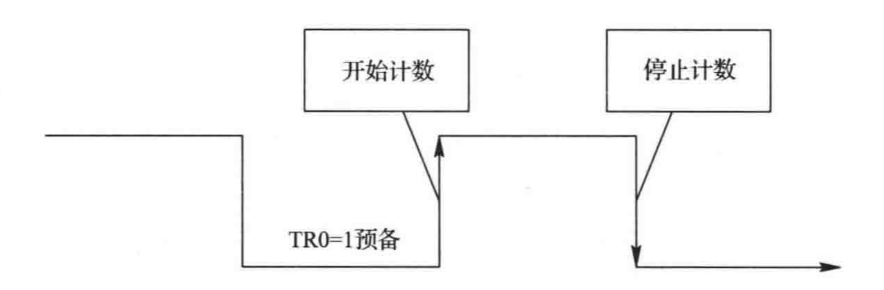
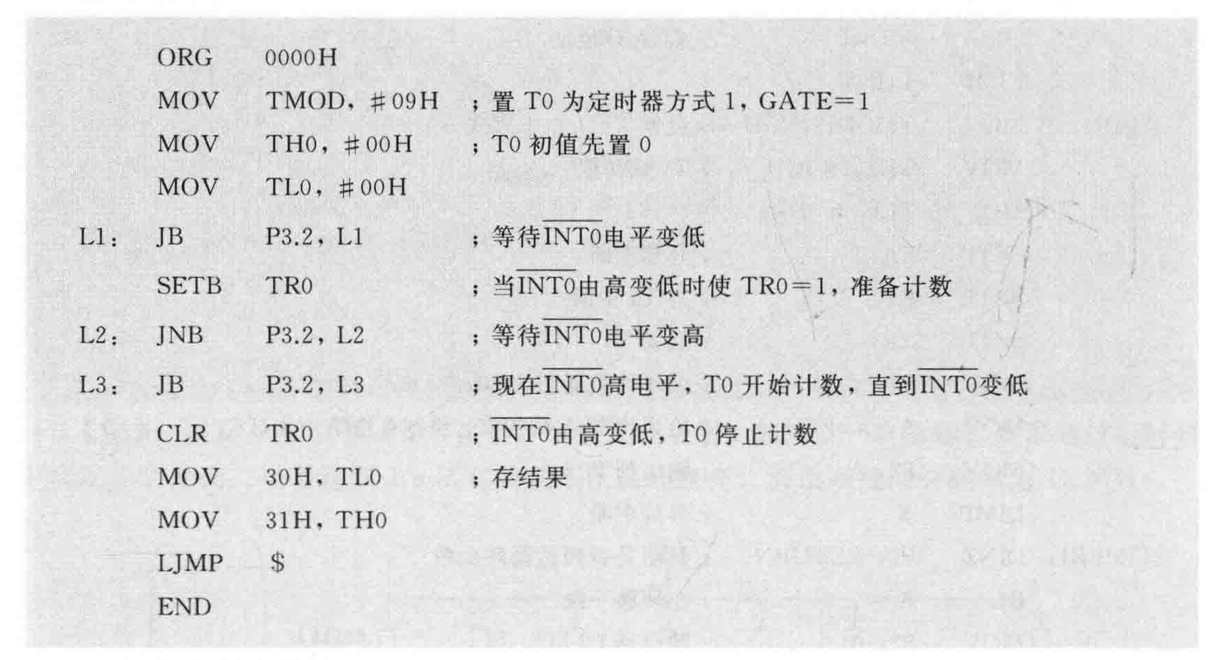


图 5-21　测量$\overline{\text{INT0}}$引脚正脉冲的处理过程

参考程序：



```
        ORG    0000H
        MOV    TMOD, #09H      ；置 T0 为定时器方式 1，GATE=1
        MOV    TH0, #00H       ；T0 初值先置 0
        MOV    TL0, #00H
L1：    JB     P3.2, L1        ；等待INT0电平变低
        SETB   TR0             ；当INT0由高变低时使 TR0=1，准备计数
L2：    JNB    P3.2, L2        ；等待INT0电平变高
L3：    JB     P3.2, L3        ；现在INT0高电平，T0 开始计数，直到INT0变低
        CLR    TR0             ；INT0由高变低，T0 停止计数
        MOV    30H, TL0        ；存结果
        MOV    31H, TH0
        LJMP   $
        END
```

4. 扩展外部中断源

MCS-51 单片机只有两个外部中断源的输入端，但实际应用中可能有两个以上的外部中断源，如果片内定时器/计数器未被使用，可以利用定时器/计数器来扩展外部中断源。

扩展方法：将定时器/计数器设置为计数模式，计数初值设定为满程，将待扩展的外部中断源接到定时器/计数器的外部计数引脚，从该引脚输入一个下降沿信号，计数器加 1 后便产生定时器/计数器溢出中断。因此，可把定时器/计数器的外部计数引脚作为扩展中断源的中断输入端。

【例 5-18】 用 T0 扩展一个外部中断源。

分析： 如果将 T0 设置为方式 2 的计数模式，TH0、TL0 的初值均为 FFH，并令 T0 允许中断、CPU 开中断，当 T0 外部引脚上出现一个下降沿信号时，TL0 计数加 1，产生溢出，将 TF0 置 1，向 CPU 发出中断请求，同时 TH0 的内容 FFH 又装入 TL0，作为下一轮的计数初值。这样，T0 引脚每输入一个下降沿，都将 TF0 置 1，向 CPU 发出中断请求，所以就相当于多了一个边沿触发的外部中断源。

参考程序：

```
MOV    TMOD, #06H
MOV    TL0, #0FFH
MOV    TH0, #0FFH
SETB   TR0          ；启动 T0 工作
SETB   EA           ；CPU 开中断
SETB   ET0          ；允许 T0 中断
⋮
```

思考与练习

1. MCS－51 单片机有哪几个中断源？CPU 响应各中断时，其中断入口地址是多少？

2. MCS－51 单片机有几个中断请求标志位？它们的清零方式各是什么？

3. MCS－51 单片机响应中断的条件是什么？

4. MCS－51 单片机外部中断源有哪两种触发方式？这两种触发方式有什么区别？

5. MCS－51 单片机外部中断源的扩展方式有哪些？

6. 简述 MCS－51 单片机定时器/计数器工作于定时和计数模式的异同点。

7. MCS－51 单片机定时器/计数器的 4 种工作方式各有什么特点？

8. 试编写一个中断初始化程序，使之允许 T0、T1，串行口中断，并且使 T1 的中断优先级最高。

9. 设 MCS－51 单片机的 f_{osc}＝12 MHz，编写程序以统计 50 s 内某外部事件发生的次数。要求利用 T1 和$\overline{INT1}$实现此功能。

10. 假设系统时钟频率为 6 MHz，利用定时器 T0 编程实现如下功能：使 P1.7 引脚上输出一个周期为 20ms 的方波。

11. 假设系统时钟频率为 12 MHz，编程实现用定时器 T1 产生定时脉冲，每隔 3 ms 从 P1.5 引脚输出脉宽为 3 个机器周期的正脉冲。

12. 假设系统时钟频率为 12 MHz，利用定时器 T1 编程实现如下功能：要求从 P2.1 引脚输出一个脉冲波形，高电平持续 3 ms，低电平持续 10 ms。

13. 假设系统时钟频率为 12 MHz，改写例 5－16 的程序，使之变成通过定时器 0 实现 8 个发光二极管每隔 2 s 从左向右依次循环点亮。

14. 假设系统时钟频率为 12 MHz，利用定时器 T0 设计两个不同频率的方波，P1.0 输出频率为 200 Hz，P1.1 输出频率为 100 Hz。

15. 假设系统时钟频率为 12 MHz，利用定时器 T0 编程实现如下功能：要求从 P1.7 引脚输出一个脉冲频率为 2 kHz、占空比为 7∶10 的脉冲宽度调制(PWM)信号。

第6章　单片机串行通信接口

串行通信是单片机应用系统与其他计算机(包括单片机)沟通的手段。本章讲述串行通信的基本概念、串行口的内部结构和工作原理，重点讲述串行通信的应用。

6.1　串行通信的基本概念

随着计算机系统的广泛应用和计算机网络的快速发展，通信功能显得越来越重要。通信既包括计算机与外部设备之间的信息交换，也包括计算机与计算机之间的信息交换。

在计算机系统中，CPU和外部通信有两种基本方式：并行通信与串行通信，图6－1是这两种通信方式的示意图。

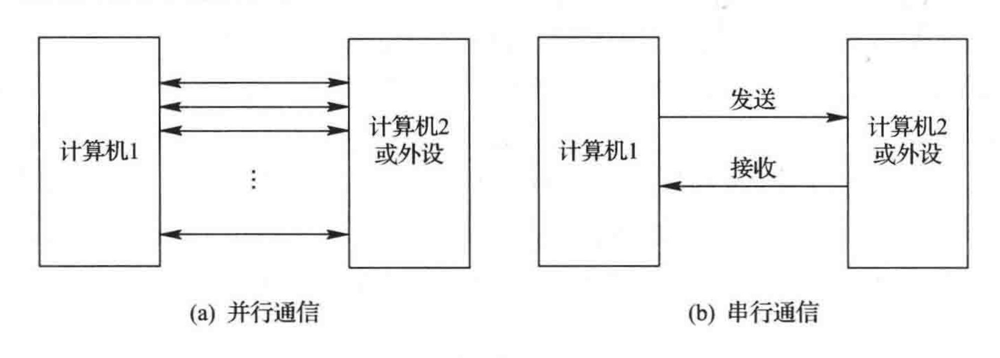


图6－1　通信基本方式

1. 并行通信

并行通信传输中有多个数据位，同时在两个设备之间传输，传送速度快，如图6－1(a)所示。并行通信主要用于近距离通信，其优点是传输速度快，处理简单。计算机内的总线结构就是并行通信的例子。

2. 串行通信

在串行通信中，数据是一位一位地在通信线上传输的，先由发送设备将并行数据经并/串转换电路转换成串行方式，再逐位经传输线传输到接收设备中，并在接收端将数据从串行方式重新转换成并行方式，以供接收方使用，如图6－1(b)所示。串行通信的数据传输速度要比并行通信的慢得多，但所需传输线少，特别适用于远距离通信。

6.1.1　串行通信的分类

按照串行通信设备的时钟控制方式，串行通信可分为异步通信和同步通信两类。

1. 异步通信

在异步通信中，数据是以字符为单位传输的，字符间隔不固定。字符帧由发送端一帧

一帧地发送，每一帧数据均为低位在前，高位在后，通过传输线被接收端一帧一帧地接收。发送端和接收端可以由各自独立的时钟来控制数据的发送和接收，这两个时钟彼此独立，互不同步。

在异步通信中，接收端是依靠字符格式来判断发送端是何时开始发送、何时结束发送的。字符格式是异步通信的一个重要指标。字符格式的规定使双方能够将同一个由“0”和“1”组成的串理解成同一种意义。在没有数据通信的情况下，发送线为高电平，每当接收端检测到传输线上发送过来的低电平(字符帧中的起始位)时，就知道发送端已开始发送数据；每当接收端接收到字符帧中的停止位时，就知道一帧字符信息已发送完毕。

在异步通信中，通信双方必须规定相同的字符帧格式和波特率，波特率大小由用户根据实际情况选定。图 6－2 是串行异步通信的字符帧格式示意图。

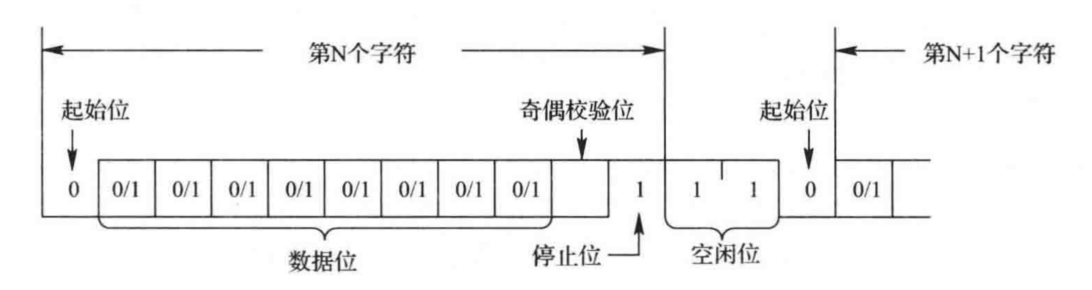


图 6－2　异步通信的字符帧格式

1) 字符帧格式

字符帧也称为数据帧，由起始位、数据位、奇偶校验位、停止位四部分组成。

① 起始位：位于字符的开头，1 位，用低电平 0 表示，表示字符的开始，通知接收端准备接收。

② 数据位：紧跟在起始位之后，可以是 5～8 位数据，发送时低位在前，高位在后。

③ 奇偶校验位：1 位，位于数据位之后，也可以没有校验位。

④ 停止位：位于字符最后，以高电平 1 表示字符的结束，告诉接收端本帧数据发送完毕，为下一帧数据做准备。

在串行通信中，两个字符之间可以没有空闲，但如果在第一个字符停止位后不是要紧接着传送下一个字符，可在停止位后加若干个“空闲位”，使线路处于等待状态。空闲位用高电平 1 表示。图 6－2 就表示了带有 2 个空闲位的字符格式。接收端不断检测线路的状态，若连续为 1 后又检测到一个 0，就知道有一个新的字符已经到来。

2) 波特率

在串行通信中，用“波特率”来描述数据的传输速率。**波特率是单位时间内传输码元符号的个数(传符号率)**，是衡量串行数据速度快慢的重要指标。比特率是单位时间内传输二进制代码的位数，其单位为 b/s(bit per second，也有用 bps 表示)。在单片机的串行通信中，波特率和比特率是相同的。国际上规定了一个标准波特率系列：110 b/s、300 b/s、600 b/s、1200 b/s、1800 b/s、2400 b/s、4800 b/s、9600 b/s、14.4 kb/s、19.2 kb/s、28.8 kb/s、33.6 kb/s、56 kb/s。例如 9600 b/s，指每秒传送 9600 位，包含字符的数位和其他必须的数位(如奇偶校验位等)。大多数串行口电路的接收波特率和发送波特率可以分

别设置，但接收方的接收波特率必须与发送方的发送波特率相同。

异步通信不需要传送同步时钟，字符长度不受限制，因此设备简单；其缺点是字符中因包含起始位和停止位而降低了有效数据的传输速率。

2. 同步通信

在异步通信中，每个字符要用起始位和停止位作为字符开始和结束的标志，占用了时间，当数据较多的时候更明显，所以在传递数据块时，为了提高速度，常去掉这些标志，采用同步通信方式。

同步通信时，将许多字符组成一个信息组，这样，字符可以一个接一个地传输，但是，在每组信息(通常称为帧)的开始要加上同步字符，当没有信息要传输时，要填上空字符，因为同步通信不允许有间隙。在同步通信过程中，一个字符可以对应5～8位。当然，对同一个传输过程，所有字符对应同样的数位，比如n位。这样，传输时，按每n位划分为一个时间片，发送端在一个时间片中发送一个字符，接收端则在一个时间片中接收一个字符。

同步通信时，一个信息帧中包含许多字符，每个信息帧用同步字符作为开始，一般情况下，同步字符和空字符使用同一个代码。在整个系统中，由一个统一的时钟控制发送端的发送和空字符。接收端能识别同步字符，当接收端检测到有一串数位和同步字符相匹配时，就认为开始一个信息帧，于是，把此后的数位作为实际传输信息来处理。同步信息帧通常由同步字符、数据字符和校验字符CRC三部分组成，格式如下：

同步字符 1	同步字符 2	数据字符 1	数据字符 2	…	数据字符 n	CRC1	CRC2

由于数据块传递开始要用同步字符来指示，同时要求由时钟来实现发送端与接收端之间的严格同步，故硬件较复杂。

6.1.2 串行通信的数据传输方式

串行通信双方进行数据传送时，根据同一时刻数据流的方向可分为三种基本的数据传送方式：单工通信、半双工通信和全双工通信。

(1) 单工通信：通信双方之间只有一根数据传输信号线，信息传送只能在一个方向上进行，如图6-3(a)所示。

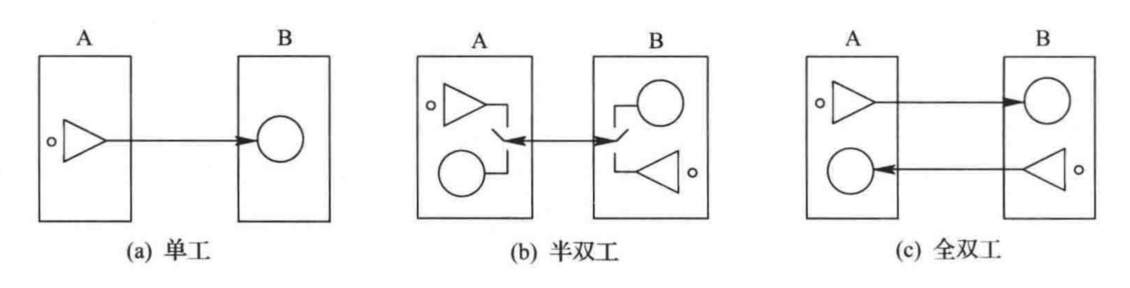


图6-3 串行通信的数据传输方式

(2) 半双工通信：通信双方之间只有一根数据传输信号线，通过接收和发送转换开关，使得双方可以交替进行发送和接收，但两个方向的数据传送不能同时进行，如图6-3(b)

所示。

(3) 全双工通信：通信双方之间有两条数据传输信号线，可以在同一时刻进行两个方向的数据传送，此时通信系统的每一端都应该设置发送器和接收器，如图 6-3(c)所示。

6.2 串行口的内部结构和工作原理

在串行通信中，数据是逐位按顺序传送的，而计算机内部的数据是并行的，因此当计算机向外设发送信息时，必须将并行数据转换成串行数据再通过串行口进行传送。反之，又必须将串行数据转换成并行数据再输入到计算机中。**能够完成异步通信的硬件电路称为 UART，即通用异步收发器**，它的主要功能是将计算机内部传送过来的并行数据转换为输出的串行数据流，将计算机外部传送过来的串行数据转换为字节；在输出数据流中加入起始位和停止位，并从接收数据流中删除起始位和停止位。

MCS-51 单片机内部具有一个采用 UART 工作方式的全双工的串行通信接口。该接口不仅可以同时进行数据的接收和发送，也可以作为同步移位寄存器使用。该接口有四种工作方式，其中字符格式有 8 位、10 位、11 位，并可以以不同的波特率工作。

6.2.1 串行口的内部结构

MCS-51 单片机内部的串行口是全双工的，其内部结构如图 6-4 所示。该接口有两根串行通信传输线 RxD(P3.0)和 TxD(P3.1)，用于数据的接收和发送。当单片机串行口和其他设备的串行口连接时，一定是**本机的接收连接对方的发送，本机的发送连接对方的接收，并且双方的地相连**(三线制)，如图 6-5 所示。

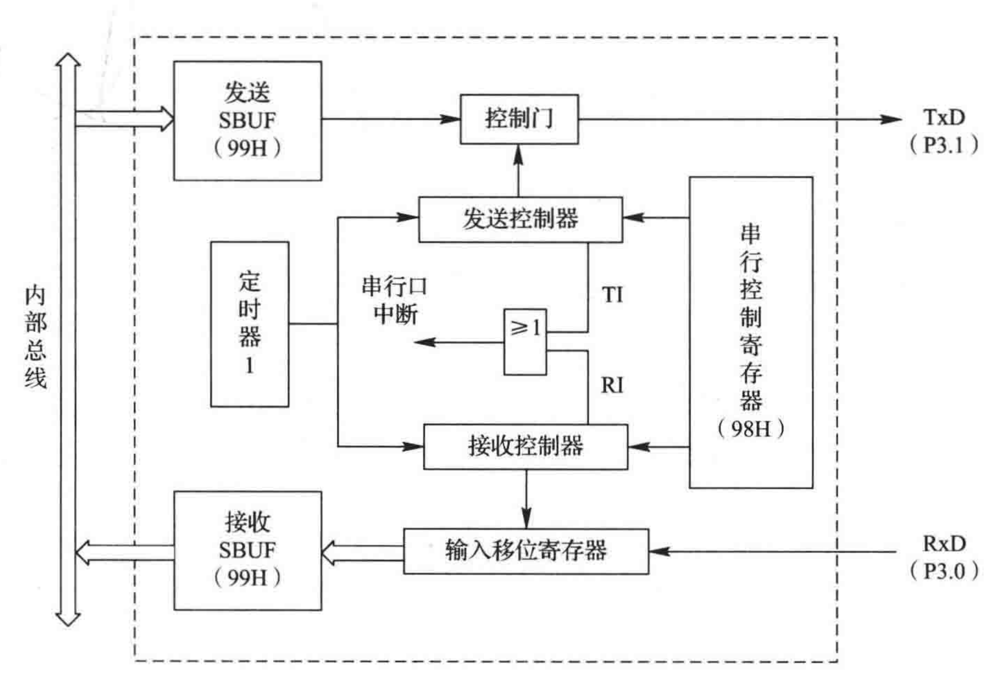


图 6-4 串行口的内部结构

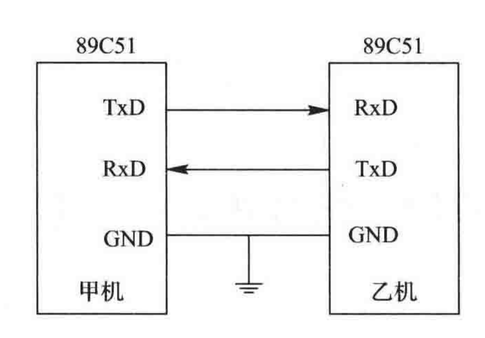


图 6-5　双机通信连接示意图

6.2.2　串行口的工作原理

MCS-51 单片机在物理上存在两个互相独立的接收、发送数据缓冲器，这样可以同时进行数据的接收和发送，实现全双工传送。发送缓冲器只能写入不能读出，接收缓冲器只能读出不能写入。串行口还有接收缓冲作用，即从接收寄存器中读出前一个已收到的字节之前就能开始接收第二个字节。两个串行口数据缓冲器(实际上是两个寄存器)通过特殊功能寄存器 SBUF 来访问。**写入 SBUF 的数据存储在发送缓冲器，用于串行发送；从 SBUF 读出的数据来自接收缓冲器，用于串行接收。**两个缓冲器共用一个地址 99H(特殊功能寄存器 SBUF 的地址)，由于接收和发送使用不同的指令，所以共用地址不会造成混淆。

发送数据需要执行以 SBUF 为目的操作数的指令，读出数据需要执行以 SBUF 为源操作数的指令，例如：

```
MOV    SBUF, A      ; (A)→(SBUF) 串行发送
MOV    A, SBUF      ; (SBUF)→(A) 串行接收
```

此外，与串行口有关的 SFR 还有 SCON 和 PCON，分别控制串行口的工作方式、工作过程以及波特率。

6.2.3　串行口的控制与状态

1. 串行口控制寄存器 SCON

MCS-51 单片机串行通信的方式选择、接收和发送控制以及串行口的状态标志均由特殊功能寄存器 SCON 控制和指示。SCON 的格式如下：

	D7	D6	D5	D4	D3	D2	D1	D0
(98H)	9FH	9EH	9DH	9CH	9BH	9AH	99H	98H
SCON	SM0	SM1	SM2	REN	TB8	RB8	TI	RI

① SM0、SM1：指定串行口的工作方式。串行口有四种工作方式，各种方式的区别在于功能、数据格式和波特率的不同，见表 6-1。

表 6-1 串行口的工作方式

SM0 SM1	工作方式	说 明	波 特 率
0 0	方式 0	同步移位寄存器	$f_{osc}/12$
0 1	方式 1	10 位异步收发	由定时器控制
1 0	方式 2	11 位异步收发	$f_{osc}/32$ 或 $f_{osc}/64$
1 1	方式 3	11 位异步收发	由定时器控制

② SM2：多机通信控制位，主要用于方式 2 和方式 3 中。

在方式 2 和方式 3 处于接收状态时，如 SM2=1，REN=1，且接收到的第 9 位数据 RB8 是 1，RI(接收中断标志位)才被置 1；若接收到的第 9 位数据 RB8 是 0，则 RI 不会置 1。

在方式 2 和方式 3 处于接收状态时，如 SM2=0，REN=1，无论接收到的第 9 位数据 RB8 是 0 还是 1，RI 都会被置 1。

在方式 1 中，如 SM2=1，只有在接收到有效停止位时，RI 才会被置 1。所以，方式 1 中 SM2 一般设置为 0，以免丢失数据。

在方式 0 中，SM2 必须为 0。

③ REN：允许串行接收控制位，由软件置 1 或清 0。REN=1，允许接收，启动串行口的 RxD，开始接收数据；REN=0，禁止接收。

④ TB8：在方式 2 和方式 3 时，它就是要发送的第 9 个数据位，一般是程控位，由软件置 1 或清 0。如在多机通信中，TB8 用于表示地址帧或数据帧。在方式 0 和方式 1 中，此位不用。

⑤ RB8：接收数据位 8。在方式 2 和方式 3 时，它就是接收到的第 9 个数据位。在方式 1 中，若 SM2=0，则 RB8 是接收到的停止位；在方式 0 中，此位不用。

⑥ TI：发送中断请求标志位。在方式 0 中，当发送完第 8 位数据时，由硬件置 1；在其他方式中，在发送停止位前，由硬件置 1。TI=1 时，申请中断，CPU 响应中断后，发送下一帧数据。在任何方式中，TI 都必须由软件清 0。

⑦ RI：接收中断请求标志位。在方式 0 中，接收第 8 位结束时，由硬件置 1；在其他方式中，在接收停止位的中间时刻，由硬件置 1。RI=1 时，申请中断，要求 CPU 取走数据。但在方式 1 中，SM2=1 时，若未接收到有效的停止位，则不会对 RI 置位。在任何方式中，RI 都必须由软件清 0。

2. 电源控制寄存器 PCON

电源控制寄存器 PCON 的格式如下：

	D7	D6	D5	D4	D3	D2	D1	D0
(87H)								
PCON	SMOD	—	—	—	GF1	GF0	PD	IDL

其中，D7 位 SMOD 是串行口波特率倍增位。SMOD 为 1 时，串行口方式 1、方式 2、方式 3 的波特率加倍。具体值见各种工作方式下的波特率计算公式。

6.2.4 串行口的工作方式

MCS-51单片机的串行口有四种工作方式，用户可以通过SCON中的SM0、SM1位来选择，见表6-1。**实际应用中，方式1和方式3应用最多。**

1. 方式0

串行口的工作方式0为移位寄存器输入/输出方式，可外接移位寄存器，以扩展I/O口，也可外接同步输入/输出设备。

方式0时，收发的数据为8位，低位在前，高位在后；波特率固定为$f_{osc}/12$，其中f_{osc}为单片机的晶振频率；数据从RxD端出入(注意，不只是接收)，这时TxD端仅输出同步移位脉冲。

1) 接收过程

CPU在每个机器周期都会采样每个中断标志。在RI=0且REN=1(允许接收)时，启动一次接收过程。这时RxD为数据输入端，TxD端输出同步脉冲，串行口以$f_{osc}/12$的波特率接收RxD引脚上的数据信息。当接收完毕时，置中断标志RI=1，发出中断申请，CPU查询到RI=1或响应中断后，通过执行以SBUF为源操作数的指令(如"MOV A，SBUF")，把SBUF中的数据送入目的操作数(如累加器A)，最后由软件复位RI。

2) 发送过程

数据的发送是执行以SBUF为目的操作数的指令(如"MOV SBUF，A")开始的，在TI=0时，8位数据按低位至高位顺序由RxD端输出，同时由TxD端输出移位脉冲，且每个脉冲输出1位数据。8位数据输出结束时，TI被置位。

以方式0工作时，SM2位(多机通信制位)必须为"0"。

2. 方式1

串行口工作于方式1时，被定义为10位的异步通信接口，即传送的一帧信息为10位二进制数——1位起始位"0"，8位数据位(先低位后高位)，1位停止位"1"，其中起始位和停止位是在发送时自动插入的，数据位由TxD发送，由RxD接收。

方式1的波特率是可变的，此时，串行口和定时器1是有关系的。在硬件电路上，T1的计数输出不仅使TF1置位，而且会产生一个脉冲送入串行口。这时方式1的波特率就取决于T1(注意只是T1，不是T0)的溢出频率(每秒钟T1溢出多少次)和PCON中的SMOD的值。

$$\text{方式1的波特率}=\left(\frac{2^{SMOD}}{32}\right)\times \text{T1的溢出频率}$$

1) 发送过程

CPU执行任何一条以SBUF为目的操作数的指令后，就启动发送。串行口的UART自动在8位数据的前后分别插入1位起始位和1位停止位，构成1帧信息。在本机内发送移位脉冲的作用下，数据依次由TxD端发出。在8位数据发出完毕以后，在停止位开始发送前，将TI置位。

2) 接收过程

当检测到RxD引脚上由1到0的跳变时开始接收过程，并复位内部16分频计数器，以实现同步。计数器的16个状态把1位时间等分成16份，并在第7、8、9个计数状态时

采样 RxD 的电平，因此每位数值采样三次，当接收到的三个值中至少有两个值相同时，这两个相同的值才被确认接收，这样可排除噪声干扰。如果检测到起始位的值不是 0，则复位接收电路，并重新寻找另一个 1 到 0 的跳变。

当检测到起始位有效时，在 REN=1 且 RI=0 的情况下，就启动接收器。在本机内接收移位脉冲的作用下，串行口把数据一位一位地移入接收移位寄存器中，直到 9 位数据全部接收齐(包括一位停止位)。接收完一帧的信息后，在 RI=0 且 SM2=0 或接收到的停止位为“1”的前提下，将接收移位寄存器中的 8 位数据送入接收缓冲寄存器 SBUF 中；接收到的停止位装入 SCON 中的 RB8，并将 RI 置位。

3. 方式 2 和方式 3

方式 2 和方式 3 为 9 位异步通信接口，串行口发送/接收的一帧信息为 11 位二进制数——1 位起始位“0”，8 位数据位(先低位后高位)，1 位奇偶校验位或其他数据位，1 位停止位“1”。方式 2 和方式 3 的发送、接收过程是完全一样的，只是波特率不同。**方式 3 的波特率是可变的**，计算公式与方式 1 的相同。

1) 发送过程

和方式 1 相似，只不过发送的一帧信息共 11 位，附加的第 9 位数据 D8 是 SCON 中的 TB8，在 8 位数据和 TB8 位发送完毕后，将中断标志位 TI 置 1。

2) 接收过程

与方式 1 基本相同，不同之处是方式 2 和方式 3 存在真正的附加的第 9 位数据 D8，共需要接收 9 位有效数据(方式 1 只是把停止位作为第 9 位处理)，D8 装入 SCON 中的 RB8。接收完一帧的信息后，在 RI=0 且 SM2=0 或接收到的第 9 位数据为“1”的前提下，将接收移位寄存器中的 8 位数据送入接收缓冲寄存器 SBUF 中；接收到的第 9 位数据装入 SCON 中的 RB8，并将 RI 置 1。

6.3 串行通信的应用

利用 MCS - 51 单片机的串行口，可以进行两个单片机之间的串行异步通信，也可以在多个单片机之间进行串行异步通信，还可以在单片机和 PC 之间进行串行通信；应用串行口可以进行数据通信，也可以传输现场采集的监测数据。总之，串行通信接口有着极其广泛的应用。

6.3.1 串行口波特率的确定和初始化

1. 波特率的计算

1) 方式 0 的波特率

方式 0 的波特率的计算公式如下：

$$波特率=\frac{f_{osc}}{12} \qquad (6-1)$$

式中：f_{osc}为晶振频率。

2) 方式 2 的波特率

方式 2 的波特率的计算公式如下：

$$波特率=\left(\frac{2^{SMOD}}{64}\right)\times f_{osc} \quad (6-2)$$

式中：SMOD 为波特率加倍位；f_{osc}为晶振频率。因为 SMOD 的值不是 0 就是 1，所以方式 2 的波特率也只有两种可能：$f_{osc}/32$ 或 $f_{osc}/64$。

3）方式 1 或方式 3 的波特率

对于串行口的方式 1 和方式 3，它们的波特率是可变的，其大小主要取决于定时器 1 的溢出频率。

波特率由定时器/计数器 T1 自动产生，通常将 T1 设置在方式 2 定时模式，并启动 T1 开始工作即可。当 T1 用作波特率发生器时，为了避免溢出而产生不必要的中断，应使 ET1=0(即不允许 T1 产生中断)。波特率的计算公式如下：

$$波特率=2^{SMOD}\times\frac{f_{osc}}{12\times 32\times(256-X)} \quad (6-3)$$

式中：X 为 T1 的初值；SMOD 为波特率加倍位；f_{osc}为晶振频率。

波特率应采用标准系列。考虑到串行通信的可靠性，MCS-51 单片机的标准波特率一般采用：110 b/s、300 b/s、600 b/s、1200 b/s、1800 b/s、2400 b/s、4800 b/s、9600 b/s、19 200 b/s。

通常是已知波特率，求 T1 的初值 X。其计算公式如下：

$$X=256-2^{SMOD}\times\frac{f_{osc}}{12\times 32\times 波特率} \quad (6-4)$$

X 值有时不是整数，这时应调整 SMOD 的取值，以选取最接近整数的值。当时钟频率选用 11.0592 MHz 时，计算出来的 X 值为整数，极易获得标准的波特率，所以很多单片机系统选用这个看起来“怪”的晶振频率就是这个道理。

2. 串行口初始化的步骤

在使用串行口前，应对相关的控制寄存器进行初始化，主要内容如下：

(1) 确定串行口工作方式(配置 SCON 寄存器)。

(2) 确定 T1 的工作方式(配置 TMOD 寄存器)。

(3) 设置 SMOD 位(若不用 SMOD，可跳过此步)。

(4) 根据波特率计算 T1 的初值，装载 TH1 和 TL1。

(5) 启动 T1(置位 TR1)。

(6) 串行口中断设置(配置 IE、IP 寄存器)。

对 T1 的初始化一般只进行一次，不能重复，否则程序运行时可能会出现错误。

【例 6-1】 某 8051 单片机控制系统，主振频率为 12 MHz，要求串行口发送数据为 8 位、波特率为 1200 b/s，编写串行口的初始化程序(设 SMOD=1)。

分析：设 SMOD=1，则 T1 的时间常数 X 的值为

$$\begin{aligned}X&=256-2^{SMOD}\times\frac{f_{osc}}{12\times 32\times 波特率}\\&=256-2^{1}\times\frac{12\times 10^{6}}{12\times 32\times 1200}\\&=256-52.08=203.92\approx 204=CCH\end{aligned}$$

参考程序：

```
MOV    SCON，#40H      ；串行口工作于方式 1
ORL    PCON，#80H      ；SMOD=1
MOV    TMOD，#20H      ；T1 工作于方式 2，定时方式
MOV    TH1，#0CCH      ；设置时间常数初值
MOV    TL1，#0CCH
SETB   TR1             ；启动 T1
```

需要指出的是，在波特率的设置中，SMOD 位数值的选择直接影响着波特率的精确度。例如波特率=2400 b/s，f_{osc}=6 MHz，这时 SMOD 可以选为 1 或 0。由于对 SMOD 位数值的不同选择，所产生的波特率误差是不同的。

若选择 SMOD=1，则

$$X=2^8-2^1\times\frac{6\times10^6}{384\times2400}=242.98\approx243$$

实际产生的波特率及误差分别为

$$波特率=2^{SMOD}\times\frac{f_{osc}}{12\times32\times(2^8-X)}=2^1\times\frac{f_{osc}}{12\times32\times(256-243)}=2403.85\ b/s$$

$$波特率误差=\frac{2403.85-2400}{2400}\times100\%=0.16\%$$

若选择 SMOD=0，则

$$X=2^8-2^0\times\frac{6\times10^6}{384\times2400}=249.49\approx249$$

实际产生的波特率及误差分别为

$$波特率=2^0\times\frac{6\times10^6}{12\times32\times(256-249)}=2232.14\ b/s$$

$$波特率误差=\frac{2400-2232.14}{2400}\times100\%=6.99\%$$

上面的分析计算说明了 SMOD 值虽然可以任意选择，但在某些情况下它会使波特率产生误差，因而在设置波特率时，需考虑 SMOD 值的选取。

表 6-2 列出了常用波特率的设置方法。

表 6-2 常用波特率的设置方法

波特率	f_{osc}/MHz	SMOD	定时器 1		
			C/$\overline{T}$	方式	重新装入值
方式 0 最大：1 Mb/s	12	×	×	×	×
方式 2 最大：375 kb/s	12	1	×	×	×
方式 1、方式 3：62.5 kb/s	12	1	0	2	FFH
19.2 kb/s	11.0592	1	0	2	FDH
9.6 kb/s	11.0592	0	0	2	FDH
4.8 kb/s	11.0592	0	0	2	FAH
2.4 kb/s	11.0592	0	0	2	F4H
1.2 kb/s	11.0592	0	0	2	E8H

6.3.2 串行口用于扩展并行I/O口

1. 用方式0扩展并行输出口

MCS-51单片机的串行口在方式0时外接一个串入并出的移位寄存器，就可以扩展一个并行输出I/O口。所用的移位寄存器应该带有输出允许控制端，这样可以避免在数据串行输出时引起并行输出端出现不稳定的输出。图6-6(a)所示为89C51串行口方式0发送的基本连线方法，所用芯片为CMOS的8位移位寄存器CD4094，其STB为输出允许控制端。当STB=1时，打开输出控制口，实现并行输出。

在串行口发送数据时，可以由TI置位后引起中断请求，并在中断服务程序中发送下一组数据；也可以通过查询TI的值来实现数据的传送(但应预先关闭中断)。

【例6-2】 用89C51串行口外接CD4094扩展8位并行输出口。如图6-6(b)所示，8位并行输出口的各位都接一个发光二极管。要求发光二极管从左到右以一定延时轮流显示，且不断循环显示。发光二极管为共阴极接法。

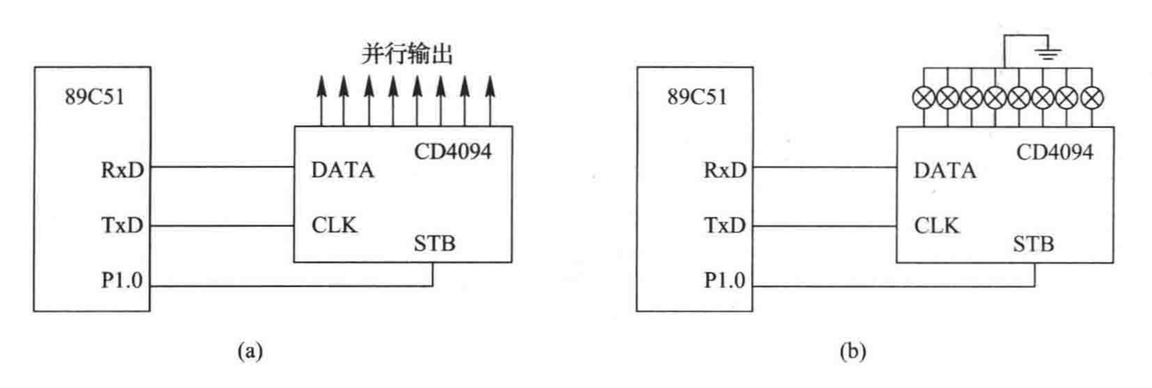


图6-6 串行口方式0扩展并行输出口

分析：设数据串行发送采用查询方式，显示的延时依靠调用延时子程序DELAY来实现，延时子程序参考前面章节内容。

参考程序：

```
        ORG     0000H
        MOV     SCON, #00H      ；串行口方式0初始化
        MOV     A, #80H         ；最左边的发光二极管先亮
        CLR     P1.0
START:  MOV     SBUF, A
LOOP:   JNB     TI, LOOP        ；查询TI
        SETB    P1.0            ；启动并行输出
        LCALL   DELAY
        CLR     TI
        RR      A               ；循环右移一位
        CLR     P1.0
        LJMP    START
        END
```

2. 用方式 0 扩展并行输入口

用方式 0 外接一个并入串出的移位寄存器就可扩展一个并行的输入口。所用的移位寄存器必须带有预置/移位控制端，由单片机的一个输出端加以控制，以实现先由 8 位输入口置入数据到移位寄存器，然后再串行移位，从而实现由单片机的串行口到接收缓冲器的传送，最后将数据由接收缓冲器读入 CPU。

图 6－7(a)所示为 89C51 串行口方式 0 接收的基本连线方法，所用芯片为 CD4014，其 $P/\overline{S}$ 端为预置/移位控制端。当 $P/\overline{S}=1$ 时，并行置入数据；当 $P/\overline{S}=0$ 时，开始串行移位。

在串行接收时，由 RI 引起中断或通过对 RI 查询来决定何时将接收到的字符读到 CPU 中(在采用查询方式时，也需预先关闭中断)。

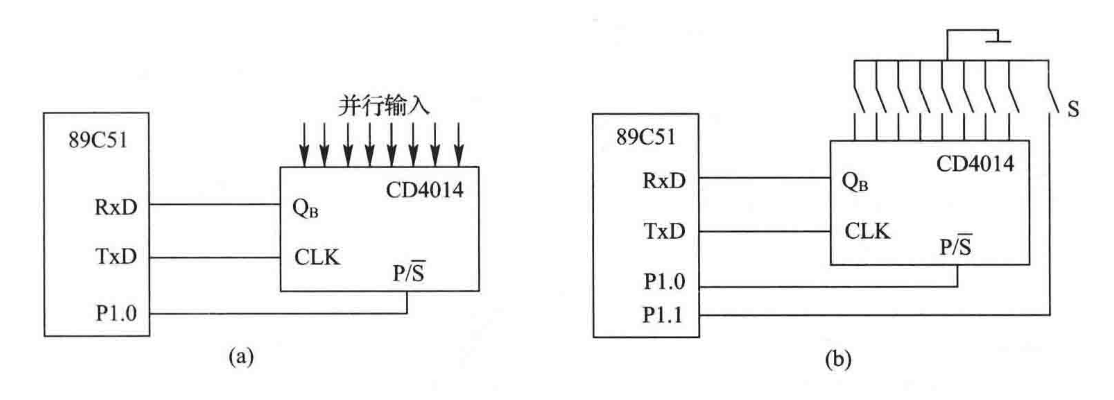
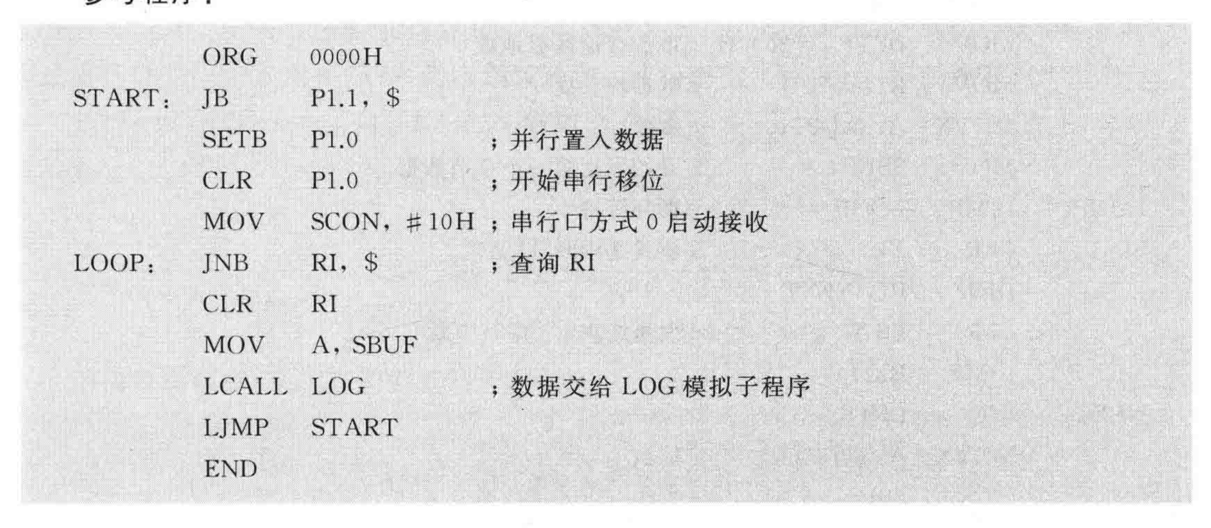


图 6－7　串行口方式 0 扩展并行输入口

【例 6－3】 用 89C51 串行口外加移位寄存器扩展 8 位并行输入口。如图 6－7(b)所示，输入数据由 8 个开关提供，另有一个开关 S 提供联络信号。当 S=0 时，表示要求输入数据。输入的 8 位开关量为逻辑模拟子程序 LOG 提供输入信号。

分析：串行口方式 0 的接收，要用 SCON 寄存器中的 REN 位来作开关控制，采用查询 RI 的方式来决定数据传送。

参考程序：



```
        ORG    0000H
START:  JB     P1.1, $
        SETB   P1.0         ;并行置入数据
        CLR    P1.0         ;开始串行移位
        MOV    SCON, #10H   ;串行口方式 0 启动接收
LOOP:   JNB    RI, $        ;查询 RI
        CLR    RI
        MOV    A, SBUF
        LCALL  LOG          ;数据交给 LOG 模拟子程序
        LJMP   START
        END
```

6.3.3 双机通信

进行双机通信时，两机应设定相同的数据格式和相同的波特率。双机异步通信的编程

通常采用两种方法：查询方式和中断方式。

单片机在上电时，发送缓冲器为空，而发送中断请求标志位 TI 初值也为“0”，这时 TI 不会变成“1”。串行口通信发送程序的开始，必须先发送一个字符，才能使得 TI 置“1”，查询方式或中断方式得以进入正常工作状态。

下面通过示例来介绍这两种方法。

【例 6-4】 按图 6-5 连接两个单片机系统(假设已经扩展了外部数据存储器)，不考虑校验，编程将甲机片外 RAM 2000H～201FH 单元中的数据块通过串行口发送到乙机片内 RAM 的 30H～4FH 单元中。假设串行口工作在方式 1，波特率为 1200 b/s，晶振频率为 12 MHz。

分析：设 SMOD=0，则 T1 的时间常数 X 的值为

$$X=256-2^{SMOD}\times\frac{f_{osc}}{12\times32\times\text{波特率}}=256-2^{0}\times\frac{12\times10^{6}}{12\times32\times1200}$$

$$=256-26.04=229.96\approx230=E6H$$

甲机发送程序(采用中断方式)：

```
        ORG     0000H
        LJMP    START
        ORG     0023H
        LJMP    SINT            ；串口中断服务程序入口
START： MOV     SP，＃60H       ；设置堆栈指针，便于中断发生时的现场保护
        MOV     SCON，＃40H     ；设置串口工作于方式 1
        MOV     TMOD，＃20H     ；设置 T1 工作于方式 2
        MOV     TL1，＃0E6H
        MOV     TH1，＃0E6H     ；赋计数初值，产生 1200 b/s 波特率
        SETB    EA              ；开总中断
        SETB    ES              ；开串口中断
        SETB    TR1             ；启动 T1
        MOV     DPTR，＃2000H   ；取数存储区首地址
        MOV     R7，＃20H       ；数据块长度
        MOVX    A，@DPTR        ；取数
        MOV     SBUF，A         ；开始发送第一个字节数据
LOOP：  LJMP    LOOP            ；循环等待
SINT：  CLR     TI              ；清发送中断 TI 请求
        DJNZ    R7，NEXT
        CLR     ES              ；发送结束，关串口中断
        LJMP    EXIT
NEXT：  INC     DPTR
        MOVX    A，@DPTR        ；取数
        MOV     SBUF，A         ；发送一个 8 位数据
EXIT：  RETI                    ；中断返回
        END
```

乙机接收程序(采用中断方式)：

```
        ORG    0000H
        LJMP   START
        ORG    0023H          ；串口中断入口地址
        LJMP   RINT
START:  MOV    SP，#60H       ；修改堆栈区指针
        MOV    SCON，#50H     ；设置串口工作于方式 1
        MOV    TMOD，#20H     ；设置 T1 工作于方式 2
        MOV    TL1，#0E6H
        MOV    TH1，#0E6H     ；赋计数初值，产生 1200 b/s 波特率
        SETB   EA
        SETB   ES             ；开中断
        SETB   TR1            ；启动 T1
        MOV    R0，#30H       ；存数存储区首地址
        MOV    R7，#20H       ；接收 32 个数
LOOP:   LJMP   LOOP           ；循环等待
RINT:   CLR    RI             ；清除接收中断请求
        MOV    A，SBUF        ；接收一个 8 位数
        MOV    @R0，A
        INC    R0             ；指向下一个字节
        DJNZ   R7，EXIT       ；判断 32 个数是否接收完
        CLR    ES             ；接收结束，关串口中断
EXIT:   RETI
        END
```

【例 6-5】 按图 6-5 连接两个单片机系统(假设已经扩展了外部数据存储器)，编程将甲机片内 RAM 40H～4FH 单元中的数据块通过串行口发送到乙机片外 RAM 的 1000H～100FH 单元中，接收过程要求判断 RB8，若出错则设 F0 标志为 1，正确则设 F0 标志为 0。假设串行口工作在方式 3，TB8 为奇偶校验位，波特率为 1200b/s，晶振频率为 12 MHz。

分析：设 SMOD=1，则 T1 的时间常数 X 的值为

$$
\begin{aligned}
X &= 256 - 2^{SMOD} \times \frac{f_{osc}}{12\times 32\times \text{波特率}} \\
&= 256 - 2^{1} \times \frac{12\times 10^{6}}{12\times 32\times 1200} \\
&= 256 - 52.08 = 203.92 \approx 204 = CCH
\end{aligned}
$$

甲机发送程序(采用查询方式)：

```
        ORG    0000H
        MOV    SCON，#0C0H    ；设置串口工作于方式 3
        MOV    TMOD，#20H     ；设置 T1 工作于方式 2
        ORL    PCON，#80H     ；SMOD=1
        MOV    TL1，#0CCH
```

```
        MOV     TH1, #0CCH          ; 赋计数初值，产生 1200 b/s 波特率
        SETB    TR1
        MOV     R1, #40H            ; 数据块首地址
        MOV     R7, #10H            ; 数据帧数
SEND:   MOV     A, @R1
        MOV     C, P                ; P 为 PSW 中奇偶校验位
        MOV     TB8, C              ; 奇偶校验位送 TB8
        MOV     SBUF, A             ; 启动发送
LOOP:   JNB     TI, LOOP
        CLR     TI
        INC     R1
        DJNZ    R7, SEND            ; 循环发送剩余数据
        LJMP    $
        END
```

乙机接收程序(采用查询方式):

```
        ORG     0000H
        MOV     SCON, #0D0H     ; 方式 3，允许接收
        MOV     TMOD, #20H      ; 设置 T1 工作于方式 2
        ORL     PCON, #80H      ; SMOD=1
        MOV     TL1, #0CCH
        MOV     TH1, #0CCH      ; 赋计数初值，产生 1200 b/s 波特率
        SETB    TR1
        MOV     DPTR, #1000H    ; 数据地址
        MOV     R7, #10H        ; 数据个数
READ:   JNB     RI, READ
        CLR     RI
        MOV     A, SBUF         ; 接收一个数
        JNB     PSW.0, PZERO
        JNB     RB8, ERROR      ; P=1 但 RB8=0 错误
        LJMP    RIGHT
PZERO:  JB      RB8, ERROR      ; P=0 但 RB8=1 错误
RIGHT:  MOVX    @DPTR, A        ; 校验正确
        INC     DPTR
        DJNZ    R7, READ
        CLR     PSW.5           ; 校验正确，F0 清 0
        LJMP    LOOP
ERROR:  SETB    PSW.5           ; 校验错误，F0 置 1
LOOP:   LJMP    LOOP
        END
```

【例 6-6】 编程实现单片机的双工通信，同时发送和接收，即每 1 s 的间隔发送片外

数据 RAM f_buffer 中的数据块(发送数据帧)，长度 f_len 不超过 100 字节，当每个数据帧发送结束后，设置一个标记位；连续接收以 $ 结束的若干组数据(接收数据帧，长度不超过 100 字节)，存入片外数据存储区 r_buffer 中，当每个数据帧接收结束后，设置一个标记位，接收数据帧的长度存入片内 RAM 的 40H 单元。设波特率为 2400 b/s，晶振频率为 12 MHz。

分析： 设 SMOD=0，则 T1 的时间常数 X 的值为

$$
\begin{aligned}
X &= 256 - 2^{SMOD} \times \frac{f_{osc}}{12 \times 32 \times \text{波特率}} \\
&= 256 - 2^{0} \times \frac{12 \times 10^{6}}{12 \times 32 \times 2400} \\
&= 256 - 52.08 = 203.92 \approx 204 = CCH
\end{aligned}
$$

考虑到收发存储区都是片外的，DPTR 是公用的，必须单独设置两个字节的指针存储单元，来保存收发数据帧当前的 DPTR。每发送或接收一个完整的数据帧后，需将收发数据帧标记置位，由主程序清 0，再进行下一组数据帧的发送和接收。

由于收发同时进行，所以必须采用中断方式，而且每次只能收发一个字节。接收和发送每一个字节时，均需要先判断收标记，再判断 RI，然后判断发标记，最后判断 TI；接收或发送完成后，还需要对 DPTR 进行保存，判断数据帧是否发送或接收完成。

每 1 s 发送一帧，采用 T0 来做定时器，在定时中断服务程序中设定数据帧发送允许。因为 $T_{cy} = 12/f_{osc} = 1\ \mu s$，考虑到定时时间较长，只能选择方式 1，最大计数值是 65 536，最大定时时间是 65 536×1 μs=65.536 ms。取 50 ms 定时，循环 20 次获得 1 s。计数初值是

$$
X = 2^{16} - \frac{50\ ms}{1\ \mu s} = 65\ 536 - 50\ 000 = 15\ 536 = 3CB0H
$$

参考程序：

```
f_len       EQU     R7              ；伪指令，定义发送数据帧计数器
r_len       DATA    R6              ；伪指令，定义接收数据帧计数器
            ORG     0100H
r_buffer:   DS  100          ；定义接收数据存储区，片外，初始地址为 0100H，长度为 100 个字节
            ORG     0300H
f_buffer:   DS  100          ；定义发送数据存储区，片外，初始地址为 0300H，长度为 100 个字节
            ORG     0000H           ；主程序入口
            LJMP    MAIN
            ORG     000BH           ；T0 入口
            LJMP    TIME
            ORG     0023H           ；串行口入口
            LJMP    COMM
MAIN:       MOV     SCON，#50H      ；串口方式 1
            MOV     TMOD，#21H      ；T0 方式 1 定时模式，T1 方式 2 定时模式
            MOV     TL0，#0B0H
            MOV     TH0，#3CH       ；T0 赋初值，定时 50 ms
            MOV     R4，#20         ；循环计数器初值 20，50 ms×20=1 s
```

```
        MOV     TL1, #0CCH
        MOV     TH1, #0CCH          ; T1 赋初值，波特率=2400 b/s
        SETB    EA                  ; 开总中断
        SETB    ES                  ; 开串口中断
        SETB    TR0                 ; 启动 T0
        SETB    TR1                 ; 启动 T1
        CLR     7FH                 ; 发送数据帧标记位，"0"为允许发送，"1"为禁止发送
        CLR     7EH                 ; 接收数据帧标记位，"0"为接收完成，"1"为未完成
M10:    JB      7EH, M20            ; 判断接收数据帧标记(由串口中断服务程序清 0)，
                                    ; 判断是否接收完
        MOV     DPTR, #r_buffer     ; 接收存储区的首地址给 DPTR
        MOV     32H, DPH            ; 接收存储区首地址高 8 位
        MOV     33H, DPL            ; 接收存储区首地址低 8 位
        MOV     r_len, #00H         ; 接收数据帧长度指针置"0"
        SETB    7EH                 ; ="1"，则允许接收下一个数据帧
M20:    ⋮                           ; 处理接收数据帧及其他
        LJMP    M10
COMM:   NOP                         ; 收发中断服务程序
        CLR     EA                  ; 屏蔽其他中断
        JNB     7EH, CO20           ; 判断接收数据帧标记
        JNB     RI, CO30            ; 先判断接收请求标志位
        CLR     RI
        MOV     A, SBUF             ; 接收一个字节数据
        MOV     DPH, 32H            ; 获取接收帧 DPTR 指针
        MOV     DPL, 33H
        MOVX    @DPTR, A            ; 将数据存入接收存储区
        INC     r_len               ; 接收数据帧计数器加"1"
        CJNE    A, '$', CO10        ; 判断"$"符号，也可以和"#24H"比较
        LJMP    CO20
CO10:   INC     DPTR                ; 指向下一个单元
        MOV     32H, DPH            ; 保存接收帧 DPTR 指针
        MOV     33H, DPL
        LJMP    CO30
CO20:   CLR     7EH                 ; 接收数据块帧标记置"0"，表明完成数据帧接收
CO30:   JNB     7FH, CO40           ; 判断发送数据帧标记
        JNB     TI, CO40            ; 再判断发送请求标志位
        MOV     DPH, 30H            ; 获取发送数据帧 DPTR 指针
        MOV     DPL, 31H
        MOVX    @DPTR, A            ; 将数据读到 A 中
        MOV     SBUF, A             ; 发一个字节
        INC     DPTR                ; 指向下一个单元
        MOV     30H, DPH            ; 保存发送数据帧 DPTR 指针
```

```
        MOV    31H, DPL
        DJNZ   f_len, CO40          ; 接收数据帧计数器减"1", 判断是否发完整个数据帧
        CLR    7FH                  ; 发送数据帧标记置"0", 表明完成数据帧发送
CO40:   SETB   EA
        RETI
TIME:   NOP                         ; 定时中断服务程序
        CLR    EA                   ; 屏蔽其他中断
        DJNZ   R4, TI10
        ⋮                           ; 给发送缓冲区存入新的发送数据帧
        MOV    DPTR, #f_buffer      ; 发送存储区的首地址给 DPTR
        MOV    30H, DPH             ; 发送存储区首地址高 8 位
        MOV    31H, DPL             ; 发送存储区首地址低 8 位
        MOV    f_len, #100          ; 发送字节数送入发送数据帧计数器
        SETB   7FH                  ; ="1", 则允许发送下一个数据帧
        MOV    R4, #20              ; 置循环计数器初值为 20, 50 ms×20=1 s
TI10:   MOV    TL0, #0B0H
        MOV    TH0, #3CH            ; T0 重赋初值, 定时 50 ms
        RETI
        END
```

6.3.4 多机通信

单片机的多机通信通常采用主从式。对于多机通信方式，要有一台主机和多台从机。主机发送的信息可以传送到各个从机或指定的从机，各从机发送的信息只能被主机接收，从机之间不能直接进行通信。

1. 多机通信原理

图 6-8 所示是多机通信的一种连接示意图。

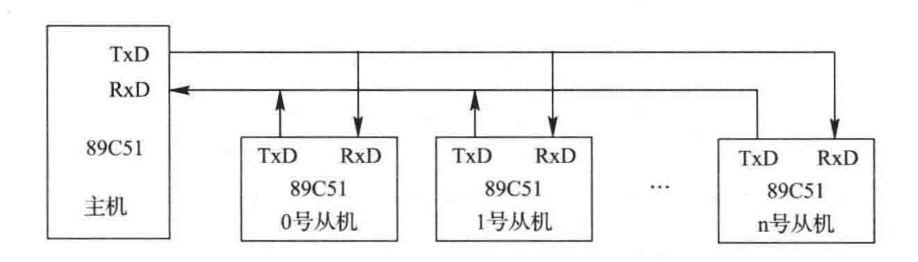


图 6-8 多机通信连接示意图

主机向从机发送的信息分地址和数据两类，以第 9 位数据作区分标志，为 0 时表示数据，为 1 时表示地址。多机通信的实现，主要依靠主、从机之间的正确设置，判断 SM2 及判断发送或接收的第 9 位数据(TB8 或 RB8)来完成。当串口工作在方式 2、3 接收时，只有同时满足下列两个条件，才会将接收到的数据装入 SBUF 和 RB8，并产生置位 RI 的信号，否则接收到的数据信号将会丢失。

① RI=0。

② SM2=0 或接收到的第 9 位数据=1。

· 若 SM2＝0，则接收到的第 9 位数据无论是 1 还是 0，数据都装入 SBUF，并置 RI＝1，向 CPU 发出中断请求。

· 若 SM2＝1，则接收到的第 9 位数据＝1，此时数据也能够装入 SBUF，并置 RI＝1，向 CPU 发出中断请求；但如果接收到的第 9 位数据＝0，则不产生中断请求，信息将被丢失，不能接收。

多机通信正是应用了条件②的接收条件实现的，具体过程如下：

（1）全部从机均初始化为方式 2 或方式 3，SM2＝1，允许中断。

（2）主机发送要寻址的从机地址，其中 TB8＝1 表示发送的是呼叫地址帧（TB8＝0 时为数据帧）。

（3）所有从机均接收主机发送的地址，并进行地址比较。

（4）被寻址的从机确认地址后，置本机的 SM2＝0，向主机返回地址，供主机核对。

（5）核对无误后，主机向被寻址的从机发送命令，通知从机接收或发送数据。

（6）通信只能在主、从机之间进行，两个从机之间的通信需通过主机作中介。

（7）本次通信结束后，从机重置 SM2＝1，主机可再对其他从机寻址。

2. 多机通信应用实例

根据上述多机通信具体过程，下面给出一个简单的应用实例。

【例 6-7】 如图 6-8 所示，采用查询方式将主机 50H～5FH 中的数据发送给 2 号从机，2 号从机将接收到的数据放到内部 RAM 的 30H～3FH 单元中。设波特率为 1200 b/s，f_{osc}＝11.0592 MHz，预置值 TH1＝0E8H。

主机程序：

```
        ORG     0000H
        LJMP    MAIN
        ORG     0030H
MAIN:   MOV     TMOD, #20H       ; 设置 T1 工作于方式 2
        MOV     TL1, #0E8H
        MOV     TH1, #0E8H       ; 产生 1200 b/s 波特率
        SETB    TR1
        MOV     SCON, #0D8H      ; 方式 3，SM2=0，允许接收，
                                 ; TB8=1 表示发送的是呼叫地址帧
M1:     MOV     SBUF, #02H       ; 主机发送要寻址的从机地址
L1:     JNB     TI, L1
        CLR     TI               ; 发送完清除 TI
L2:     JNB     RI, L2
        CLR     RI
        MOV     A, SBUF
        XRL     A, #02H          ; 确认被寻址的从机地址
        JZ      RHT
        LJMP    M1
RHT:    CLR     TB8              ; 主机向被寻址的从机发送数据，TB8=0
                                 ; 表示发送的是数据帧
```

```
        MOV     R0, #50H
        MOV     R7, #10H
L3:     MOV     A, @R0              ；开始发送数据
        MOV     SBUF, A
L4:     JNB     TI, L4
        CLR     TI
        INC     R0
        DJNZ    R7, L3
        LJMP    $
        END
```

2 号从机程序：

```
        ORG     0000H
        LJMP    MAIN
        ORG     0030H
MAIN:   MOV     TMOD, #20H      ；设置 T1 工作于方式 2
        MOV     TL1, #0E8H
        MOV     TH1, #0E8H      ；产生 1200 b/s 波特率
        SETB    TR1
        MOV     R0, #30H
        MOV     R6, #10H
        MOV     SCON, #0F0H     ；方式 3，SM2=1(接收地址帧)，允许接收
SR1:    JNB     RI, SR1
        CLR     RI
SR2:    MOV     A, SBUF
        XRL     A, #02H         ；判断是否寻址本机
        JNZ     SR1
        CLR     SM2             ；确认地址后，置本机的 SM2=0
        MOV     SBUF, #02H      ；向主机返回地址
SR3:    JNB     TI, SR3
        CLR     TI
SR4:    JNB     RI, SR4
        CLR     RI
        JNB     RB8, RHT
        SETB    SM2             ；若接收的仍为地址帧，则置位 SM2，
                                ；返回，重新接收地址帧
        LJMP    SR1
RHT:    MOV     A, SBUF         ；接收数据帧
        MOV     @R0, A
        INC     R0
        DJNZ    R6, SR4
        LJMP    $
        END
```

6.3.5 单片机与 PC 之间的通信

随着计算机的广泛应用，上、下位机的主从工作方式逐渐被数据采集和控制系统所使用。单片机价格低廉、控制方便，可作为从机，进行现场数据采集和控制；PC 的计算能力强、运行速度快、人机交互便捷，可作为主机。单片机和 PC 都具有异步通信接口，但输出电平不同，单片机为 TTL 电平，PC 为 RS-232 电平，所以**单片机与 PC 之间必须通过电平转换才能实现串行通信**。

1. RS-232 接口

在计算机的主板上一般都装有异步通信适配器(也可以通过 USB 口转换)，它使计算机有能力与其他具有 RS-232 标准接口的设备进行通信。而 MCS-51 单片机本身有一个全双工的串行口，因此只要配上电平转换的驱动电路、隔离电路就可以和 RS-232 接口组成一个简单的通信通道。

RS-232C 是异步串行通信中常用的标准接口(也称为标准总线)，它是由美国电子工业协会公布的。

RS-232C 的具体规定如下：

1) 机械特性

连接器：由于 RS-232C 并未定义连接器的物理特性，因此，出现了 DB-25 和 DB-9 等各种类型的连接器，其引脚的定义也各不相同。图 6-9 所示是常用的两种连接器。

现在的 PC 上使用的 COM1 和 COM2 就是标准的 RS-232C 接口，采用 9 针结构。

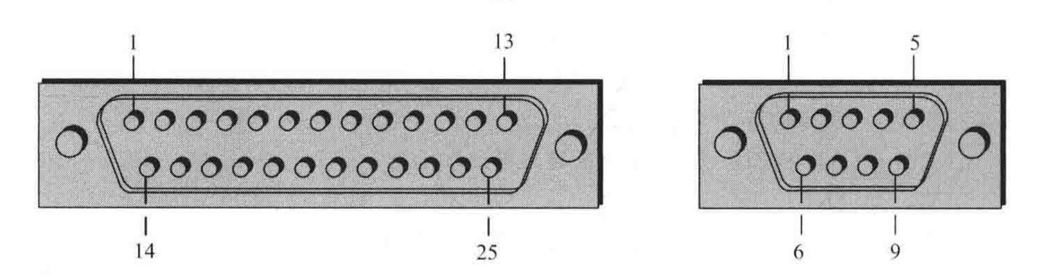


图 6-9 DB-25(阳头)连接器和 DB-9(阳头)连接器

2) 功能特性

RS-232C 接口主要信号线的功能如表 6-3 所示。

表 6-3 RS-232C 接口主要信号线的功能

引脚序号	信号名称	符 号	流 向	功 能
2(3)	发送数据	TxD	DTE→DCE	DTE 发送串行数据
3(2)	接收数据	RxD	DTE←DCE	DTE 接收串行数据
4(7)	请求发送	RTS	DTE→DCE	DTE 请求 DCE 将线路切换到发送方式
5(8)	允许发送	CTS	DTE←DCE	DCE 告诉 DTE 线路已接通可以发送数据
6(6)	数据设备准备好	DSR	DTE←DCE	DCE 准备好
7(5)	信号地			信号公共地

续表

引脚序号	信号名称	符 号	流 向	功 能
8(1)	载波检测	DCD	DTE←DCE	表示 DCE 接收到远程载波
20(4)	数据终端准备好	DTR	DTE→DCE	DTE 准备好
22(9)	振铃指示	RI	DTE←DCE	表示 DCE 与线路接通，出现振铃

注："引脚序号"一列中括号内的是 DB-9 连接器的引脚序号。

3) 电气特性

RS-232C 规定了自己的电气标准，采用负逻辑电平，即逻辑"0"为+5 V～+15 V，逻辑"1"为-5 V～-15 V。因此，RS-232C 不能和 TTL 电平直接相连，使用时必须进行电平转换。MCS-51 单片机的输入、输出电平均为 TTL 电平，而 PC 配置的是 RS-232C 标准接口，所以两者连接时要加电平转换电路。

常用的电平转换集成电路如图 6-10 所示，采用 MAX232 等芯片。仅需+5V 电源，内置电子升压泵将+5 V 转换为-10 V～+10 V，内置两个发送器和两个接收器，且与 TTL/CMOS 电平兼容，使用非常方便。

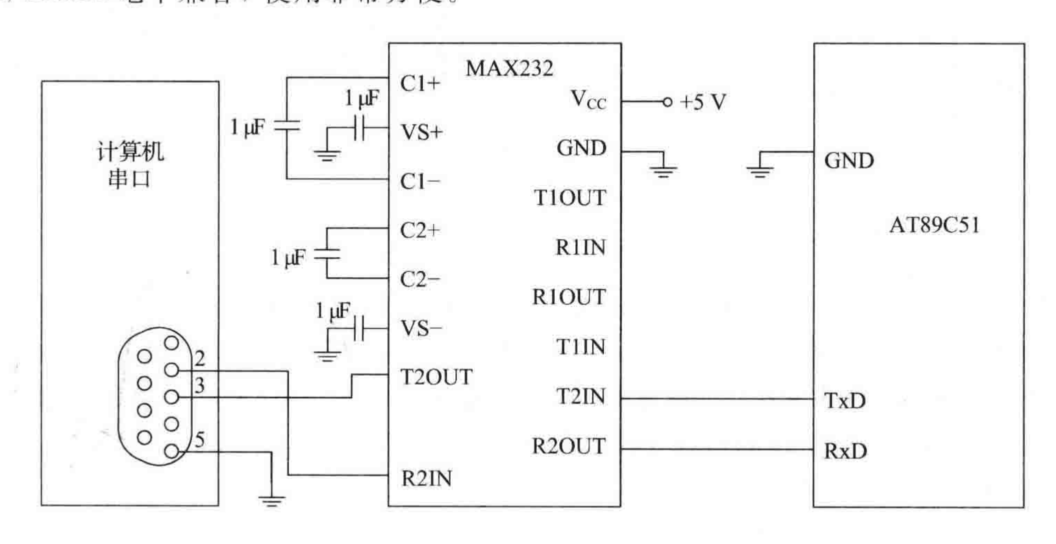


图 6-10 TTL 和 RS-232 电平转换集成电路

4) 适用范围

RS-232C 接口总线适用于通信距离仅为几十米、传输速率小于 20 kb/s 的设备。如果距离增加，可以适当地降低传输速率，以保证通信的可靠性。

5) 系统连接

用 RS-232C 总线连接系统时，有远程通信方式和近程通信方式之分。远程通信是指传输距离大于几十米的通信，其特点是在通信线路中必须使用调制解调器 Modem。如果传输距离小于几十米，则称为近程通信，其特点是通信双方可使用 RS-232C 电缆直接互连。

(1) 远程通信。如图 6-11 所示是采用 RS-232C 总线，并通过调制解调器实现计算机和一个前置数据采集器(可以由单片机实现)的远程通信的原理框图。

(2) 近程通信。当计算机与终端之间，或两台 PC 之间，或 PC 与数据采集器(可以由单片机实现)之间采用 RS-232C 总线标准近距离串行通信时，可将两个 DTE 直接连接，

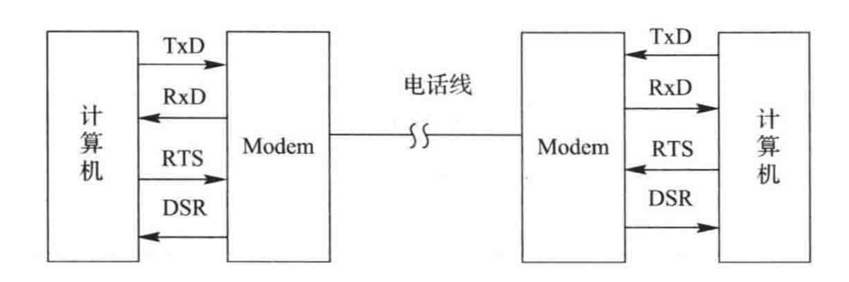


图 6-11 用调制解调器实现计算机远程通信

从而省去作为 DCE 的调制解调器。

图 6-12(a)是一种最简单的“三线制”的连接方式。在此种连接方式中，仅需将“发送数据”与“接收数据”交叉相连以及将双方的地线相连即可。此种连接方式不适用于需要检测“清除发送”、“载波检测”、“数据设备就绪”等信号状态的通信程序。

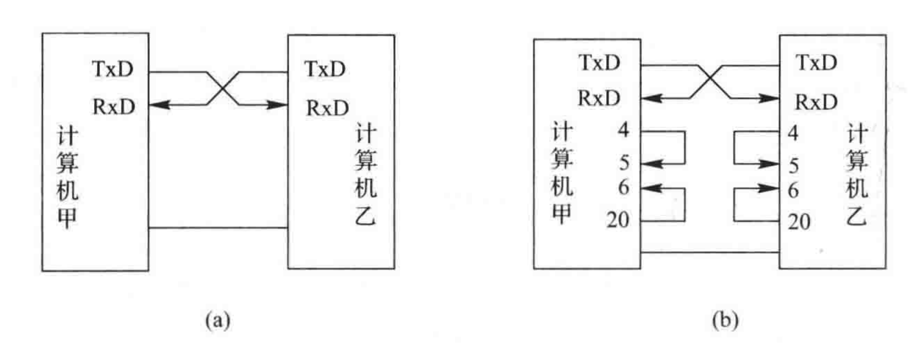


图 6-12 无联络线方式和联络线短接方式

图 6-12(b)除了按“三线制”进行连接外，还将同一设备的“请求发送”、“清除发送”和“载波检测”连在一起，将“数据终端就绪”和“数据设备就绪”连在一起。对于此种连接方式，那些需要检测“消除发送”、“载波检测”、“数据设备就绪”等信号状态的通信程序可以运行下去，但并不能真正检测到对方的状态，只是程序受到该连接方式的“蒙蔽”而已。

6) RS-232C 接口存在的问题

(1) 传输距离短、速率低。RS-232C 接口总线适用于通信距离仅为几十米、传输速率最大为 20 kb/s 的设备。

(2) 有电平偏移。RS-232C 接口收发双方共地。当通信距离较远时，两端的地电位差别较大，信号地上的地电流会产生压降，当一方输出的逻辑电平到达对方时，其逻辑电平可能偏移较大，严重时会发生逻辑错误。

(3) 抗干扰能力差。RS-232C 采用单端输入/输出，传输过程总的干扰和噪声会混在正常的信号中。为了提高信噪比，RS-232C 标准不得不采用比较大的电压摆幅。

2. RS-485 接口

针对 RS-232C 标准存在的问题，为了适用于更远的传输距离和更快的传输速率，EIA 制定了新的串行通信标准 RS-485 和 RS-422，其中 RS-485 应用广泛，通常采用的转换器件为 MAX485。图 6-13 所示为由 MAX485 构成的 RS-485 标准通信接口的示意图。

由于 MAX485 输出的高阻特性，除了能构成点对点的通信外，还能实现点对多点的通信。在多点通信系统中，必须有一个主站，其余的为从站，多个从站用地址识别。当主站

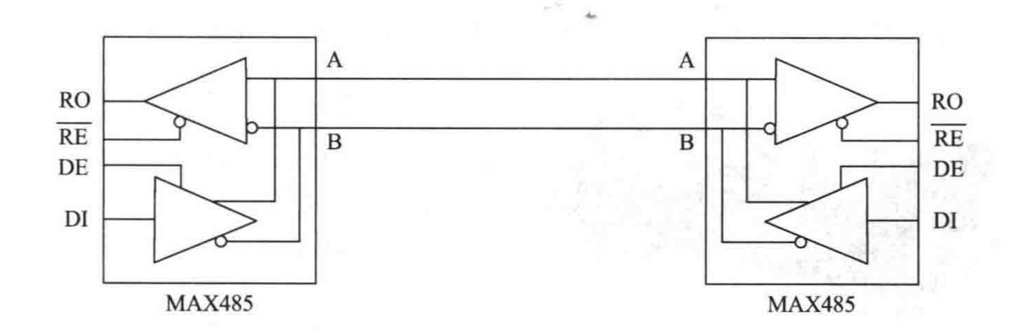


图6-13 由MAX485构成的RS-485标准通信接口

要求某一从站发送信息时，将由主站发出与该从站地址相匹配的命令，选中的从站发回信息，未选中的则不予响应。

图6-14为采用点对多点通信的分布式通信系统。该系统采用半双工的工作模式，通过控制发送允许信号和接收允许信号，来控制信号的流向。在正常工作状态下，控制主站的发送允许信号和接收允许信号使输出发送器有效，即处于发送状态；同时控制所有从站的发送允许信号和接收允许信号使接收器有效，即处于接收状态。当主站发出命令时，所有的从站接收到命令；当发送的命令字中的地址信息与某一从站的地址相匹配时，该从站使能信号控制发送器有效，即转为发送状态，地址不匹配的其余从站仍处于接收状态，以免信号线上的数据发生冲突；而主站在发送命令后转为接收状态，此时主站接收从选中的从站发回的信息，地址不匹配的其余从站对接收到的信息不予处理，从而保证了通信的正常进行。当从站发送完数据后必须转为接收状态，而主站在接收到数据后转为发送状态。

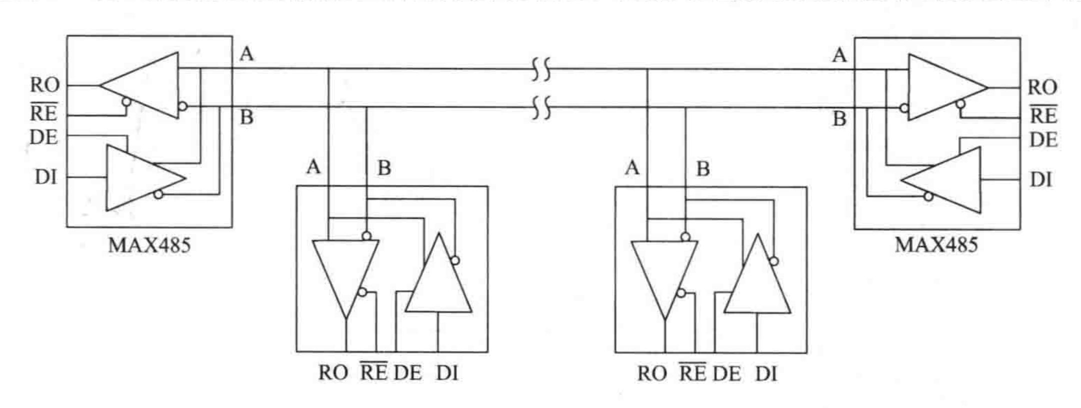


图6-14 采用点对多点通信的半双工分布式通信系统

点对多点通信时，发送器连接的点数由发送器的驱动能力和接收器的负载特性决定，通常能驱动的点数为32个，而后期推出的器件可达64、128甚至256个点。由于接收器通过接收端的电平差来判断传输的电平，因此对连接系统的导线没有严格的要求，只需采用双绞线即可构成网络。

由于PC默认的只带有RS-232接口，故有两种方法可以得到PC的RS-485电路：

① 通过RS-232/RS-485转换器(见图6-15)将PC串口RS-232信号转换成RS-485信号，对于情况比较复杂的工业环境，最好选用防浪涌带隔离栅的产品。

② 通过PCI多串口卡，可以直接选用输出信号为RS-485类型的扩展卡。

对于MAX485，只需将DI、RO分别与单片机的RxD、TxD直接相连，将MAX485的DE和

$\overline{RE}$短接并连接到单片机的某 I/O 口上，以控制 MAX485 进行收、发数据，如图 6－16 所示。

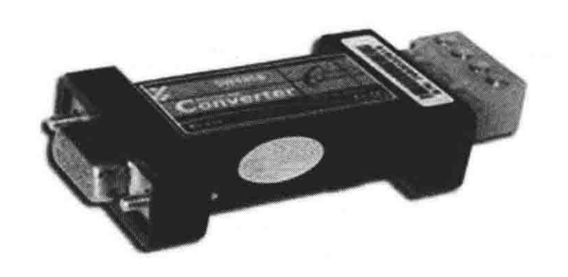


图 6－15 RS－232/RS－485 转换器

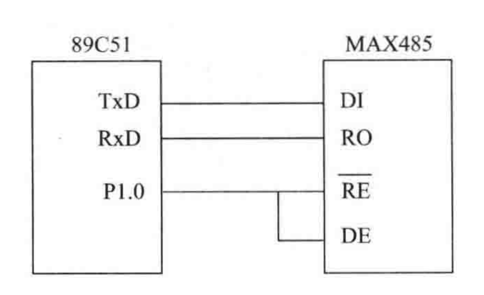


图 6－16 89C51 与 MAX485 的连接图

3. 编程举例

开发 MCS－51 单片机串行口通信程序时，开发者通常只负责单片机端的程序，怎样确认串行口通信程序是否正确呢？我们可以采用串口调试工具软件来辅助调试。

【例 6－8】 按照图 6－10 连接计算机和单片机，使用串口调试工具软件，发送一组数据，编写单片机程序，将接收到的数据再发送到计算机上，并在接收窗口显示。

(1) **计算机串口检查**：将 PC 的串行口的收、发引脚用连接线连接。运行串口调试软件(网上有多种此类软件可供下载试用)，在发送区输入发送信息，在接收区会显示出与发送信息相同的接收信息，如图 6－17 所示。



图 6－17 串口调试工具软件界面

(2) **联机编程调试**：连接单片机与 PC 的串行口，在串口调试工具软件上设置波特率：9600 b/s；数据位：8 位；停止位：1 位，编写单片机串行口测试程序，利用 PC 的串行口调试软件实现字符或字符串的发送和接收。

单片机串行口测试程序：

```
ORG     0000H
LJMP    MAIN
```

```
        ORG     0023H           ；串行口中断入口地址
        LJMP    SINT
MAIN：  MOV     TMOD，#20H      ；置定时器1，方式2，自动重载模式
        MOV     SCON，#50H      ；置串口方式1，REN=1
        MOV     TH1，#0FDH      ；波特率为9600 b/s
        MOV     TL1，#0FDH
        SETB    TR1             ；启动定时器1
        SETB    ES              ；串口中断允许
        SETB    EA              ；总中断允许
        LJMP    $
SINT：  NOP                     ；串行通信中断服务程序
SSS：   JNB     RI，SSS         ；判断接收中断请求
        CLR     RI
        MOV     A，SBUF         ；从接收缓冲区SBUF收一个字节
        NOP
        NOP
        NOP                     ；用空操作形成一个微小的延时
        MOV     SBUF，A         ；将接收到的字节再发送到串行输出缓冲区SBUF
SSSS：  JNB     TI，SSSS        ；判断发送中断请求
        CLR     TI
        RETI                    ；中断返回
        END
```

该程序运行后，在串口调试软件的发送区键入的字符会由PC的串行口发送到单片机串行口，单片机串行口接收到的这些信息再由单片机的串行口发送到PC，并由串口调试软件显示在接收区，如图6-18所示。

本例程只作为单片机串口程序调试方法加以介绍，实际的计算机端串行通信程序应根据实际需要编写。

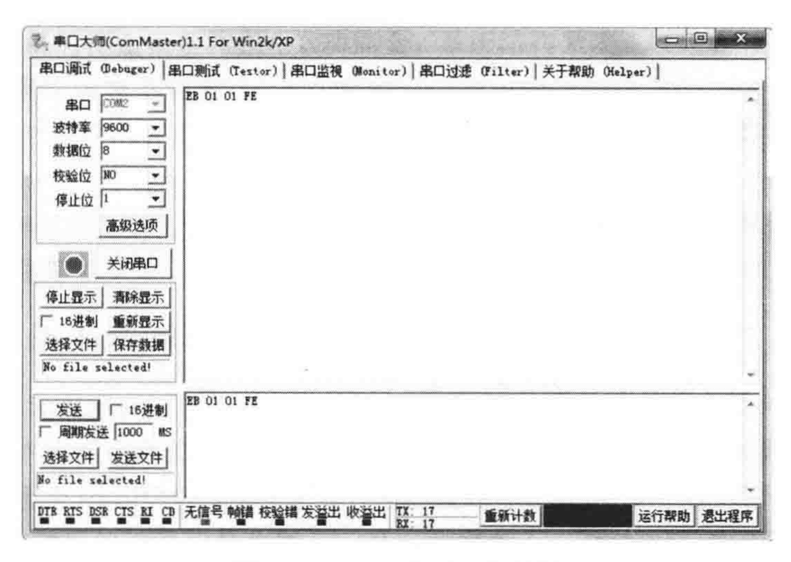


图6-18 串口调试运行结果

思考与练习

1. 什么是串行异步通信？它有哪些特点？有哪几种字符格式？

2. 定时器用作波特率发生器时，为什么要使用方式 2？若已知通信选用的波特率和系统时钟频率，如何计算 T1 的初值？

3. 串行口工作在方式 1 和方式 3 时，定时器 T1 工作于方式 2 的初值表达式是什么？

4. 某异步通信接口，其字符形式由 1 个起始位 0、8 个数据位、1 个校验位和 1 个停止位 1 组成，当该接口每分钟传送 2000 个字符时，传送波特率为多少？

5. 已知定时器 T1 设置为方式 2，用作波特率发生器，系统时钟频率为 6 MHz，求可能产生的最高和最低波特率。

6. 设计一个 MCS - 51 单片机的双机通信系统，试编程将甲机片内 30H～3FH 单元中的数据块通过串行口发送到乙机片内 RAM 的 40H～4FH 单元中，要求接收和发送均采用中断方式。设晶振频率为 11.0592 MHz，波特率为 2400 b/s。

7. 设计一个 MCS - 51 单片机的双机通信系统，试编程将甲机片外 1000H～1FFFH 单元中的数据块通过串行口发送到乙机片外 RAM 的 3000H～3FFFH 单元中，要求接收和发送均采用中断方式。设晶振频率为 11.0592 MHz，波特率为 4800 b/s。

8. 用 89C51 单片机的串行口扩展并行 I/O 接口，控制 16 个发光二极管依次发光，要求画出电路图，并编写相应的程序。

9. 改写例 6 - 7 的程序，使之变成采用查询方式将主机 40H～5FH 单元中的数据发送给 3 号从机，3 号从机将接收到的数据放到内部 RAM 的 30H～4FH 单元中。设波特率为 9600 b/s，f_{osc}＝11.0592 MHz。

第7章　单片机并行扩展技术

51系列单片机在一块芯片上集成了CPU、RAM、ROM及I/O口等计算机的基本部件，使用起来非常方便。但单片机内的RAM、ROM和I/O接口数量有限，不够使用时，需进行扩展。因此，单片机的系统扩展主要是指外接数据存储器、程序存储器或I/O接口等，以满足应用系统的需要。

本章重点讲述编址方法，以及ROM、RAM、I/O扩展的硬件电路，并在键盘和显示的内容上给出实际编程案例，最后介绍A/D和D/A转换器及应用。

7.1　单片机的最小系统

最小应用系统，是指能维持单片机运行的最简单配置的系统。这种系统成本低廉、结构简单，常用来构成简单的控制系统，如开关状态的输入/输出控制等。对于片内有ROM/EPROM的单片机，其最小应用系统即为配有晶振、复位电路和电源的单个单片机。对于片内无ROM/EPROM的单片机，其最小系统除了外部配置晶振、复位电路和电源外，还应当外接EPROM或E^2PROM作为程序存储器使用。

当然，最小系统有可能无法满足应用系统的功能要求。比如，有时即使有内部程序存储器，但由于程序很长，程序存储器容量可能不够；对一些数据采集系统，内部数据存储器容量也可能不够等，这就需要根据情况扩展EPROM、RAM、I/O口及其他所需的外围芯片。

7.1.1　80C51/89C51最小应用系统

51系列单片机的特点就是体积小，功能全，系统结构紧凑，硬件设计灵活。对于简单的应用，最小系统即能满足要求。

80C51/89C51是片内有ROM/E^2PROM的单片机，因此，用这些芯片构成的最小系统简单、可靠。

用80C51/89C51单片机构成最小应用系统时，只要将单片机接上时钟电路和复位电路即可，具体电路见第2章。

7.1.2　8031最小应用系统

8031是片内无程序存储器的芯片，因此，其最小应用系统必须在片外扩展EPROM。图7-1所示为外接程序存储器的最小应用系统。片外8 KB单元地址要求地址线13根(A0～A12)，它由P0和P2.0～P2.4组成；地址锁存器的锁存信号为ALE(Address Latch Enable)。程序存储器的选通信号为$\overline{PSEN}$(Program Store Enable)。由于程序存储器芯片只有一片，故其片选线($\overline{CE}$)可以直接接地。

图 7－1　8031 最小应用系统

8031 芯片本身的连接除了$\overline{EA}$必须接地外(选择外部存储器)，其他与 80C51/89C51 最小应用系统一样，也必须有复位及时钟电路。

7.2　总线扩展及编址方法

7.2.1　单片机的外总线结构

图 7－2 给出了单片机应用系统的外总线结构示意图。

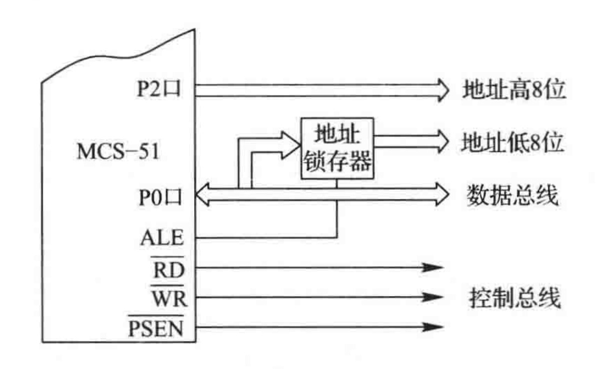


图 7－2　单片机的三总线结构

单片机是通过地址总线、数据总线和控制总线与外部交换信息的。

1. 地址总线

地址总线宽度为 16 位，因此其寻址范围为 2^{16}＝64 KB。

高8位地址由P2口提供，因为P2口具有输出锁存功能，且用于外部扩展时一般不做他用，所以不需地址锁存器。

低8位地址由P0口提供，P0口是数据、地址分时使用的通道口。为了保存地址信息，需外加地址锁存器。锁存器的锁存信号为引脚ALE输出的控制信号，在ALE的下降沿将P0口输出的地址锁存。

2. 数据总线

数据总线由P0口提供，其宽度为8位，该口为三态双向口，是应用系统中使用最为频繁的通道。单片机所有需要与外部交换的数据、指令、信息，除少数可直接通过P1口传送外，大部分都通过P0口传送。

3. 控制总线

系统扩展时，常用的控制信号如下：

(1) $\overline{EA}$：内部和外部程序存储器的选择控制信号。当$\overline{EA}=0$时，只访问外部程序存储器。

(2) ALE：地址锁存允许信号。

(3) $\overline{PSEN}$：外部程序存储器读信号。

(4) $\overline{WR}$：外部数据存储器写信号。

(5) $\overline{RD}$：外部数据存储器读信号。

其中，$\overline{EA}$引脚的电平高低由用户决定；ALE信号自动产生；$\overline{PSEN}$在访问外部程序存储器时自动产生；$\overline{RD}$、$\overline{WR}$信号在执行MOVX指令时自动产生。

7.2.2 单片机的扩展能力

根据单片机地址总线宽度，在片外可扩展的存储器最大容量为64 KB，地址范围为0000H～FFFFH。由于片外数据存储器和程序存储器的操作使用不同的指令和控制信号，允许两者的地址重复，因此片外可扩展的数据存储器与程序存储器各为64 KB。

片外数据存储器与片内数据存储器的操作指令不同，所以也允许两者的地址重复，即外部扩展数据存储器的地址可以从0000H开始。

MCS-51单片机片外程序存储器与片内程序存储器采用相同的操作指令，对片内、片外程序存储器的选择依靠硬件来实现：当$\overline{EA}=0$时，不论片内有无程序存储器，片外程序存储器的地址都可以从0000H开始；但当$\overline{EA}=1$时，前4 KB的地址(0000H～0FFFH)为片内程序存储器所有，片外扩展的程序存储器的地址只能从1000H开始设置。

为了配置外围设备而需要扩展的I/O口与片外数据存储器统一编址，不再另外提供地址线。所以，当应用系统需要大量配置外围设备以及扩展较多的I/O口时，要占去大量的RAM地址。

7.2.3 地址译码方法

扩展芯片与CPU地址总线的连接方式，必须满足对这些芯片所分配的地址范围的要求。CPU发出的地址信号必须实现两种选择：片选(即对扩展芯片的选择，使相关芯片的片选端$\overline{CS}$为有效)和字选(即在选中的芯片内部再选择某一存储单元)。片选信号和字选信号均由CPU发出的地址信号经译码产生。通常的连接方法是：将扩展芯片的地址线和单

片机的地址总线中的若干根低位地址线对应相连，其余的地址线(通常是 P2 口的高位地址)通过地址译码来产生外部扩展芯片的片选信号$\overline{CE}$或$\overline{CS}$。常用的地址译码方法有线选法和译码法两种。

1. 线选法

所谓线选法，就是将多余的高位地址线中单独的一根直接接到扩展芯片的使能端上。线选法编址的特点是简单明了，且不需要另外增加电路。由于片选线一般都是采用高位地址线，对于扩展芯片数量较多的应用系统，这些芯片所需要的片选信号的数量有可能多于可用的高位地址线数量，因此采用线选法无法解决问题。另外，这种编址方法对存储空间的使用是断续的，不能充分有效地利用存储空间，且扩充容量受限，因而只适用于小规模单片机系统的外围芯片扩展。

图 7-3 所示为线选法应用实例。图中所扩展的芯片地址范围如表 7-1 所示，其中×可以取"0"，也可以取"1"，用十六进制数表示的地址如下：

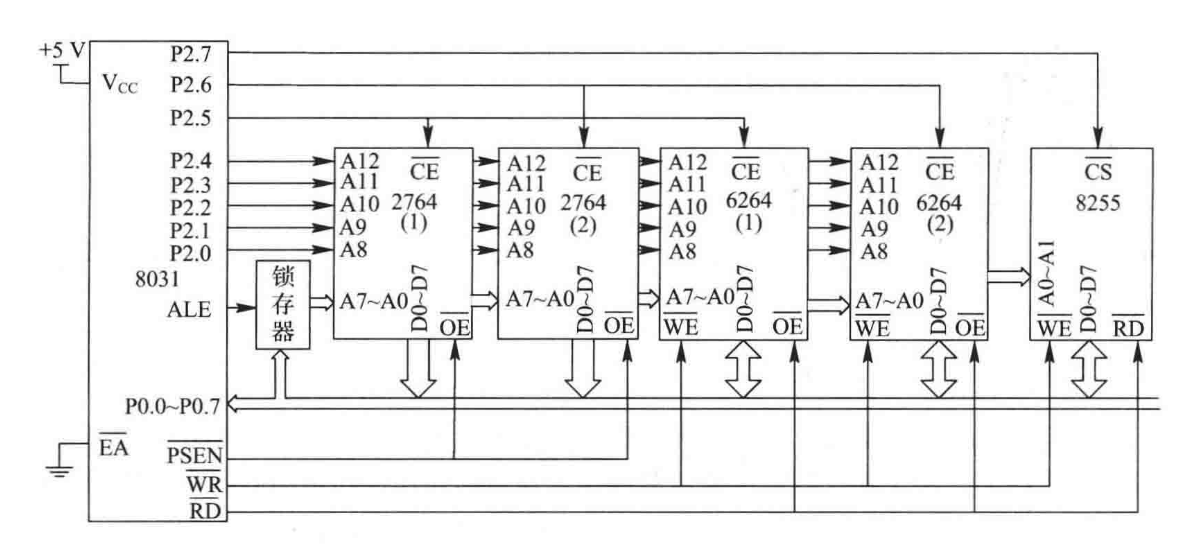


图 7-3　线选法应用实例

2764(1)：4000H～5FFFH，或 C000H～DFFFH，有地址重叠现象。

2764(2)：2000H～3FFFH，或 A000H～BFFFH，有地址重叠现象。

6264(1)：C000H～DFFFH。

6264(2)：A000H～BFFFH。

8255：6000H～6003H，或 6FFCH～6FFFH，或其他。8255 虽然只用了四个存储单元，但占用的地址空间可能从 6000H 一直到 6FFFH，具体取决于那些"×"地址线的使用情况。

表 7-1　图 7-3 的线选法地址表

地址	P2.7 A15	P2.6 A14	P2.5 A13	P2.4 A12	P2.3 A11	P2.2 A10	P2.1 A9	P2.0 A8	P0.7 A7	P0.6 A6	P0.5 A5	P0.4 A4	P0.3 A3	P0.2 A2	P0.1 A1	P0.0 A0	
2764 (1)	× ×	1 1	0 0	0 1	0 1	0 1	0 1	0 1	0 1	0 1	0 1	0 1	0 1	0 1	0 1	0 1	首址 末址

续表

地址	P2.7 A15	P2.6 A14	P2.5 A13	P2.4 A12	P2.3 A11	P2.2 A10	P2.1 A9	P2.0 A8	P0.7 A7	P0.6 A6	P0.5 A5	P0.4 A4	P0.3 A3	P0.2 A2	P0.1 A1	P0.0 A0	
2764	×	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	首址
(2)	×	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	末址
6264	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	首址
(1)	1	1	0	1	1	1	1	1	1	1	1	1	1	1	1	1	末址
6264	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	首址
(2)	1	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	末址
8255	0	1	1	×	×	×	×	×	×	×	×	×	×	×	0	0	首址
	0	1	1	×	×	×	×	×	×	×	×	×	×	×	1	1	末址

2. 译码法

所谓译码法，就是使用译码器对系统的高位地址进行译码，以译码输出作为存储芯片的片选信号，将低位地址作为存储芯片的片内地址。这是一种最常用的存储器编址方法，能有效地利用存储空间，适用于大容量多芯片存储器扩展；其缺点是增加了硬件电路。译码电路可以使用现有的译码器芯片，常用的译码芯片有 74LS139（双 2－4 译码器）和 74LS138(3－8 译码器)等。

1）全译码法

全译码法是指将各扩展芯片上的地址线均接到单片机系统的对应的地址总线上，各芯片的选择利用译码电路实现，地址译码器使用余下的全部地址线，地址与存储单元一一对应，也就是一个存储单元只占用一个唯一的地址。全译码法的特点是：各扩展芯片均有独立片选控制线，且地址连续，这种方法可以消除地址空间重叠现象和断续现象，可扩展较多的外围芯片。如图 7－4 所示是全译码法的一个简单应用实例，图中各芯片的地址范围如下：

2764(1)：000 0 0000 0000 0000B～000 1 1111 1111 1111B，0000H～1FFFH。

2764(2)：001 0 0000 0000 0000B～001 1 1111 1111 1111B，2000H～3FFFH。

6264(1)：000 0 0000 0000 0000B～000 1 1111 1111 1111B，0000H～1FFFH。

6264(2)：001 0 0000 0000 0000B～001 1 1111 1111 1111B，2000H～3FFFH。

8255：010× ×××× ×××× ××00B～010× ×××× ×××× ××11B，4000H～4003H，…，5FFCH～5FFFH 等。

2）部分译码法

部分译码法是指地址译码器仅对余下高位地址线的一部分进行译码，产生片选信号。这种方法也会产生地址重叠现象。

全译码法和部分译码法的区别在于剩余的高位地址线是否全部接地址译码器，在设计地址译码器电路时，需要充分注意。除了使用译码器进行译码，还可以使用组合逻辑电路构成译码电路，此时只需符合相应的地址安排。

随着存储器集成电路技术的进步，大容量存储器的成本和售价大幅降低。单片机的外

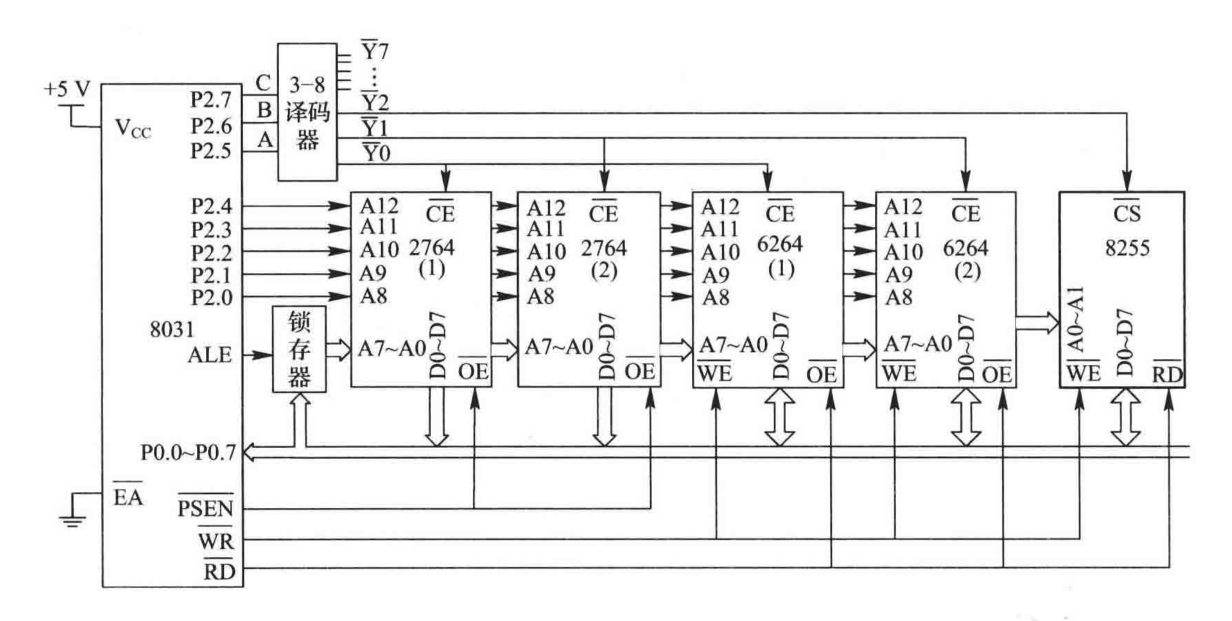


图 7-4　全译码法应用实例

扩存储器通常只要一片即可，例如使用 27512/27C512(64 KB)，其价格和 2764、2732 相当。所以，以上给出的存储器扩展方法已无多大实用意义，但介绍的总线控制技术仍然十分有用，可以有效地应用到其他外围芯片的扩展中。

7.3　存储器的扩展

7.3.1　EPROM 程序存储器的扩展

1. EPROM 芯片简介

EPROM 芯片是紫外线擦除可编程只读存储器，其上有一个玻璃窗口，在紫外线的照射下，存储器中的各位信息都为 1，一般擦除次数可达上百次，甚至可达到万次，所以称为紫外线擦除可编程只读存储器。EPROM 重新编程时，需从插座上取出，放到专门的擦写器上，擦除干净的 EPROM 芯片通过编程器将应用程序固化到芯片中。

通常采用的 EPROM 标准芯片有 2716(2 K×8 bit)、2732(4 K×8 bit)、2764(8 K×8 bit)、27128(16 K×8 bit)、27256(32 K×8 bit)、27512(64 K×8 bit)等。其中“27”是产品代号，表示是 EPROM 芯片，后面是芯片容量，单位是 Kbit，将这个数据除以 8 就是常用的字节容量值。如 2764 芯片就是 8 KB 的 EPROM 芯片，由于容量是 8 KB，因此该芯片的地址线是 13 根(2^{13}＝8 K，A0～A12)。

在上面的几种 EPROM 芯片中，2716、2732 为 24 引脚，两者兼容。如将 2732 插入 2716 芯片的插座上也可以工作，但只有 2KB 有效。2764、27128、27256、27512 芯片均为 28 引脚，均可向下兼容。同样，将 27128 芯片插入 2764 芯片的插座上也可以工作，但只有 8KB 有效。图 7-5 所示是部分 27 系列 EPROM 芯片的引脚图，各引脚功能如下：

(1) A0～Ai：地址线(不同容量芯片的地址线数目不同，i 表示芯片的地址线数目)。

(2) D0～D7：8 位输出数据线。

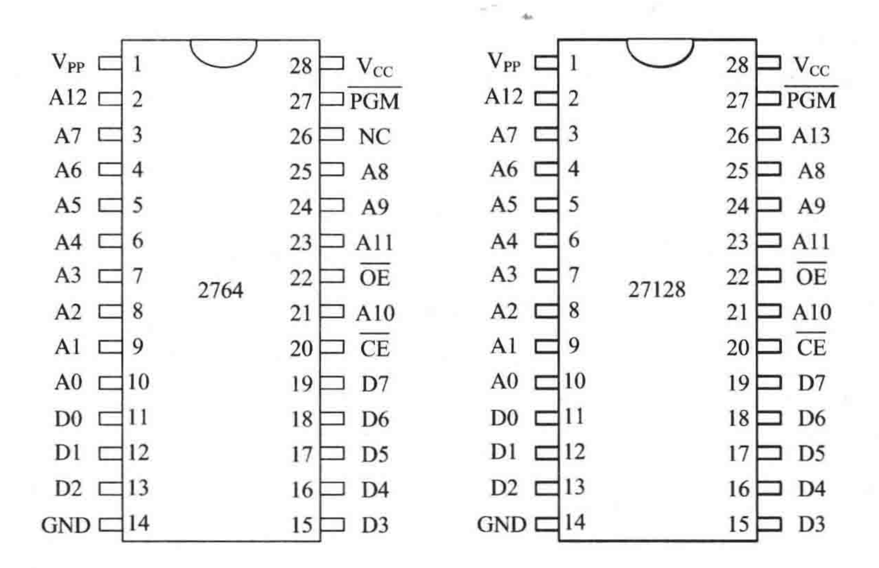


图 7－5 部分 27 系列 EPROM 芯片的引脚图

(3) $\overline{CE}$：片选端。

(4) $\overline{OE}$：输出允许端。

(5) V_{PP}：编程电压。

(6) $\overline{PGM}$：编程脉冲输入端。

2. EPROM 基本扩展法

图 7－6 给出了外部程序存储器的典型连接方法。

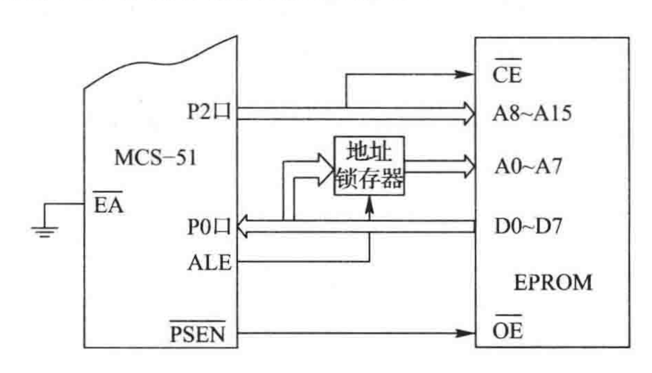


图 7－6 外部程序存储器的典型连接方法

1) 地址线的连接

存储器低 8 位地址线 A0～A7 与 P0 口(经锁存器)的输出端相连。

存储器高 8 位地址线 A8～A15 中的若干根与 P2 口(无须经过锁存器)相连。P2 口其余的高位地址线用作片选地址线(线选或译码)。

2) 数据线的连接

存储器的 8 位数据线与 P0 口相连。

3) 控制线的连接

(1) $\overline{PSEN}$：外部程序存储器读信号，与 EPROM 芯片的输出允许端$\overline{OE}$相连。

(2) ALE：接至地址锁存器锁存信号端。

(3) $\overline{EA}$：采用 8031/8032 时，$\overline{EA}$应接地；采用 8751、89C51 等型号的芯片并且使用到

内部程序存储器时，$\overline{EA}$应接高电平。

【例 7-1】 用两片 2764 EPROM 芯片为 8031 单片机扩展 16 KB 的程序存储器，试画出线路连接图。

连接方法：

（1）将两片 2764 EPROM 芯片低 8 位地址线 A0～A7 通过地址锁存器与 8031 P0 口的 P0.0～P0.7 相连，高 5 位地址线 A8～A12 直接与 P2 口的 P2.0～P2.4 相连。

（2）将两片 2764 EPROM 芯片的数据线 D0～D7 直接接到 P0 口，作为数据总线。

（3）用 P2.5 和一个“非门”产生两个片选信号，将 P2.5 直接接到上方 EPROM 的$\overline{CE}$端，另一个经过“非门”接到下方 EPROM 的$\overline{CE}$端，如图 7-7 所示。当 P2.5=0 时，选中上方的 EPROM；当 P2.5=1 时，反相选中下方的 EPROM。

两个芯片的地址范围：

在图 7-7 的连接方式中，P2.6 和 P2.7 并没有参与译码，而用 P2.5 经过“1-2”译码器产生两个片选信号，所以该连接方式属于部分译码方式，两个芯片的地址范围如下：

① 2764(1)：××00 0000 0000 0000B～××01 1111 1111 1111B，由于“×”可以取“0”、“1”之中的任一值，故十六进制地址有四组，即

0000H～1FFFH，4000H～5FFFH，8000H～9FFFH，C000H～DFFFH

② 2764(2)：××10 0000 0000 0000B～××11 1111 1111 1111B，十六进制地址有四组，即

2000H～3FFFH，6000H～7FFFH，A000H～BFFFH，E000H～FFFFH

可以看出，部分译码方式和线选法一样同样浪费地址空间，出现地址重叠的情况。

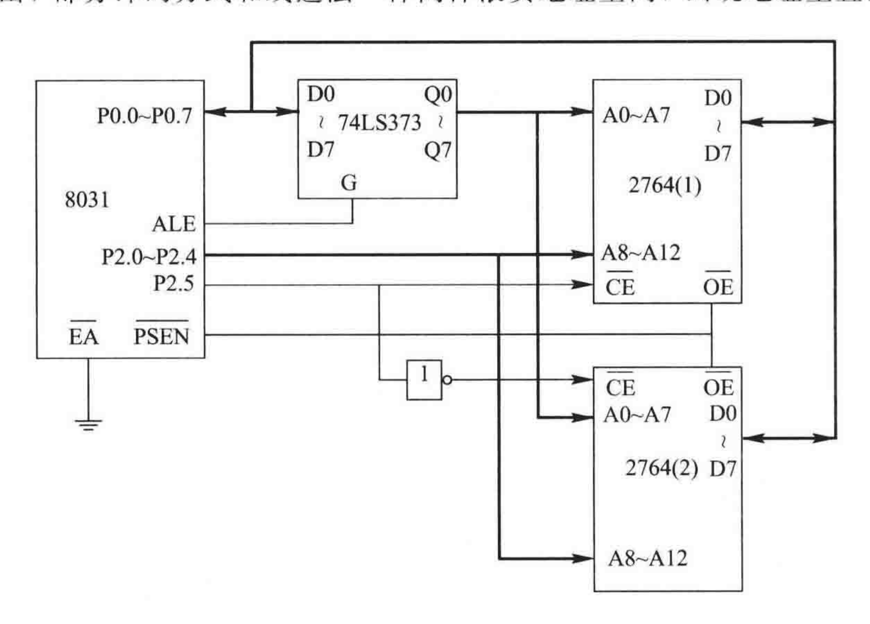


图 7-7 两片 2764 扩展 16 KB 程序存储器

7.3.2 E^2PROM 程序存储器的扩展

1. E^2PROM 芯片简介

电擦除可编程只读存储器 E^2PROM 是近年来国外厂家推出的新产品，它的主要特点

是能在计算机系统中进行在线修改，并能在断电的情况下保持修改的结果。因此，自从 E^{2}PROM问世以来，在智能化仪器仪表、控制装置、终端机、开发装置等各种领域中受到极大的重视。下面介绍典型 E^{2}PROM 存储器芯片 Intel 2864A。

Intel 2864A 是 8 K×8 bit 的电可擦除可编程只读存储器，单一+5V 供电，最大工作电流为 140 mA，维持电流为 60 mA。由于其片内设有编程所需的高压脉冲产生电路，因而无需外加编程电源和写入脉冲即可工作。Intel 2864A 采用典型的 28 引脚结构，与常用的 8 KB 静态 RAM 6264 引脚完全兼容。其芯片引脚如图 7－8 所示。

(1) A0～A12：13 位地址线。

(2) D0～D7：8 位输出/输入数据线。

(3) $\overline{CE}$：片选端。

(4) $\overline{OE}$：输出允许端。

(5) $\overline{WR}$：写允许端。

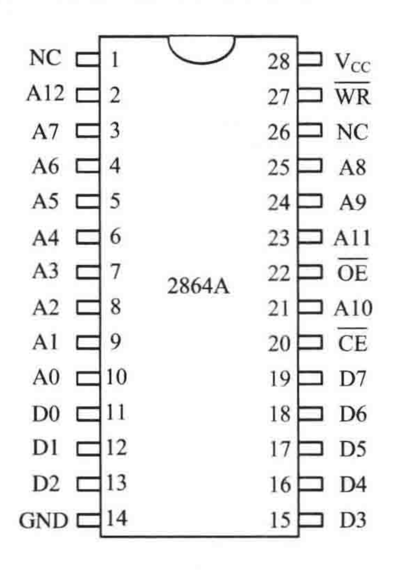


图 7－8 2864A 引脚图

2. E^{2}PROM 基本扩展法

MCS－51 单片机扩展 E^{2}PROM 时，地址线和数据线的连接方法与 EPROM 连接方法相同，控制线的连接中，只有单片机的$\overline{PSEN}$及$\overline{RD}$信号与 EPROM 的连接有所不同。若 E^{2}PROM仅作为程序存储器，则将$\overline{PSEN}$信号与$\overline{OE}$引脚相连。若 E^{2}PROM 仅作为数据存储器，则将$\overline{RD}$信号与$\overline{OE}$引脚相连。如 E^{2}PROM 既作为数据存储器用，又作为程序存储器用，可将$\overline{PSEN}$信号和$\overline{RD}$信号经过相“与”后与$\overline{OE}$引脚相连。图 7－9 所示是 E^{2}PROM 芯片的一般连接图。

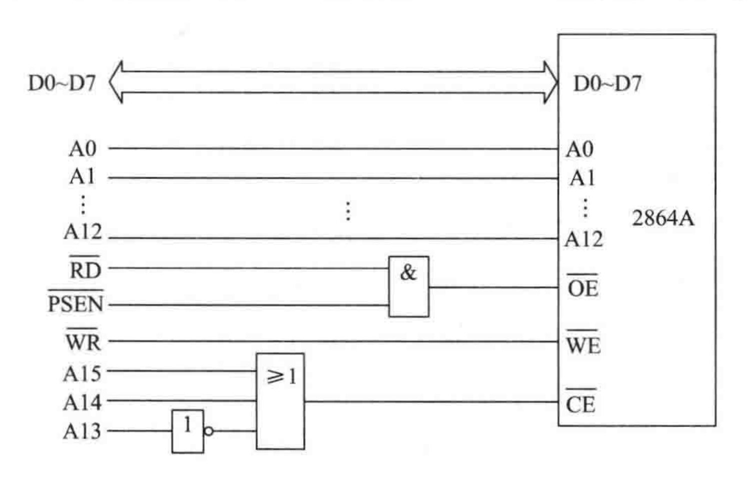


图 7－9 E^{2}PROM 芯片的一般连接图

7.3.3 数据存储器及其扩展

在 MCS－51 单片机的产品中，片内数据存储器的容量一般很小。当数据量较大时，需要在片外扩展 RAM 数据存储器，扩展的容量最大可达 64 KB。但由于 MCS－51 系列单片机对片外扩展的 I/O 口采用外部数据存储器映射方式进行输入/输出(即将片外 I/O 口的数据寄存器当作外部数据存储器的一个单元来看待，在指令系统及接口上不对这两者加

以区别)，在这种情况下，允许直接扩展的外部数据存储器的容量将不足 64 KB。

1. RAM 芯片简介

RAM 的典型芯片有 6116(2 K×8 bit)、6264(8 K×8 bit)等。图 7-10 所示为 6264 芯片引脚图，其引脚功能如下：

(1) A0～A12：13 位地址线。

(2) D0～D7：8 位输出/输入数据线。

(3) $\overline{\text{CE1}}$、CE2：片选端。

(4) $\overline{\text{OE}}$：输出允许端。

(5) $\overline{\text{WE}}$：写允许端。

(6) V_{CC}、GND：+5 V 电源和接地端。

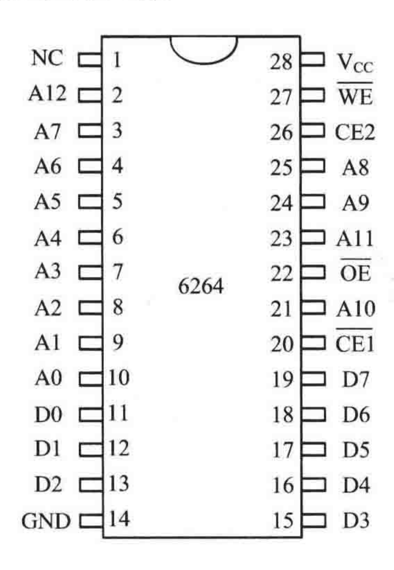


图 7-10　6264 芯片引脚图

2. 外部数据存储器的扩展方法

外部数据存储器的连接方法与程序存储器的连接方法大致相同，区别在于控制线的连接。图 7-11 所示为外部数据存储器的一般连接方法，其中$\overline{\text{WR}}$是外部数据存储器的写信号，与 RAM 芯片的$\overline{\text{WE}}$引脚相连；$\overline{\text{RD}}$是外部数据存储器的读信号，与 RAM 芯片的$\overline{\text{OE}}$引脚相连。

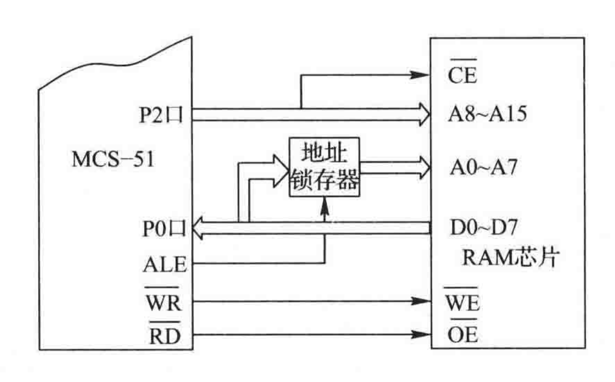
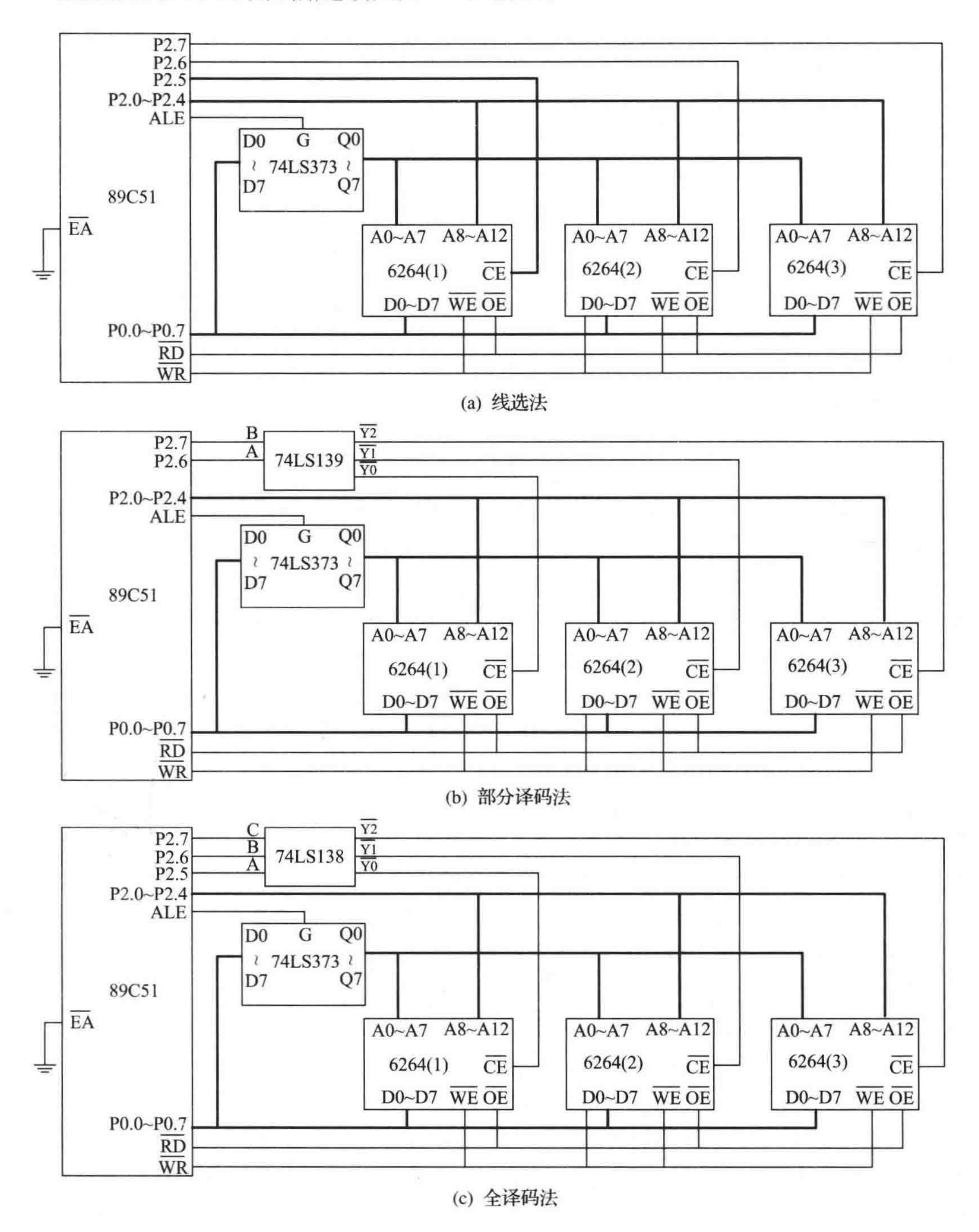


图 7-11　外部数据存储器的一般连接方法

【例 7-2】 用三片 6264 芯片为 AT89C51 单片机扩展 24 KB 的外部数据存储器，分

别采用线选法、部分译码法和全译码法来实现，并给出各芯片的地址范围，译码芯片不限。

连接方法和地址：其线路连接如图7－12所示。



(a) 线选法

(b) 部分译码法

(c) 全译码法

图7－12 用6264芯片扩展24 KB的数据存储器

地址线：将6264芯片的低8位地址线A0～A7通过地址锁存器74LS373与AT89C51 P0口的P0.0～P0.7相连，高5位地址线A8～A12直接与P2口的P2.0～P2.4相连。

数据线：将6264芯片的数据线D0～D7直接接到P0口，作为数据总线。

控制线：将 6264 芯片的输出允许端$\overline{OE}$直接接到 AT89C51 的 P3.7($\overline{RD}$)，写允许端$\overline{WE}$直接接到 P3.6($\overline{WR}$)。ALE 与锁存器 74LS373 的 G 相连，产生锁存控制信号。

(1) 线选法：在图 7-12(a)中，由于扩展三片，故将 P2.5、P2.6、P2.7 分别与三个 6264 芯片的$\overline{CE}$相连，产生三个片选信号。

单片机 P2.7=1、P2.6=1 和 P2.5=0 是固定的，仅 P2.5 是低电平，选中第一个 RAM，其余的地址线与 6264(1)地址线直接相连，可以任意为"0"或"1"，均为有效地址。6264(1)的地址范围是 1100 0000 0000 0000B～1101 1111 1111 1111B，十六进制地址范围为 C000H～DFFFH。

P2.7=1、P2.6=0 和 P2.5=1 时，选中第二个 RAM，6264(2)的十六进制地址范围为 A000H～BFFFH。

P2.7=0、P2.6=1 和 P2.5=1 时，选中第三个 RAM，6264(3)的十六进制地址范围为 6000H～7FFFH。

显然，用线选法会使存储空间不连续，也不能充分有效地利用存储空间，扩充存储容量受限。

(2) 部分译码法：在图 7-12(b)中，P2.6、P2.7 与 74LS139(2-4 译码器)输入端 A、B 相连，三个输出信号$\overline{Y0}$、$\overline{Y1}$和$\overline{Y2}$与三片存储器的$\overline{CE}$端相连，由于 P2.5 空闲，可以选择 0 或 1，这时，第一个 RAM 可访问的二进制地址范围为 00×0 0000 0000 0000B～00×1 1111 1111 1111B，由此可得两组地址：0000 0000 0000 0000B～0001 1111 1111 1111B 和 0010 0000 0000 0000B～0011 1111 1111 1111B，另两个 RAM 可用类似方法获得地址编码，所以三片 6264 芯片的十六进制地址范围如下：

6264(1)：0000H～1FFFH 和 2000H～3FFFH；

6264(2)：4000H～5FFFH 和 6000H～7FFFH；

6264(3)：8000H～9FFFH 和 A000H～BFFFH。

通过地址范围可以看出，每个芯片占用了 16 KB 的地址空间，而每个芯片实际的容量是 8 KB，原因是地址线 P2.5 没有参与译码，这种部分译码方式也存在地址空间浪费的情况。在这种情况下，地址 0000H 和 2000H 指向的是同一数据存储器单元，其余类似。

(3) 全译码法：如果把译码器换成 74LS138，则 P2.5(A13)也参与了译码，这种全译码(所有剩余高地址线都参与译码)方式不存在地址空间浪费，如图 7-12(c)所示。使用$\overline{Y0}$、$\overline{Y1}$和$\overline{Y2}$作片选信号，则三片 6264 芯片的地址范围如下：

6264(1)：000 0 0000 0000 0000B～000 1 1111 1111 1111B，0000H～1FFFH。

6264(2)：001 0 0000 0000 0000B～001 1 1111 1111 1111B，2000H～3FFFH。

6264(3)：010 0 0000 0000 0000B～010 1 1111 1111 1111B，4000H～5FFFH。

注意：在扩展单片数据存储器时，存储器片选端能否直接接地，还需考虑应用系统中有无 I/O 口及外围设备扩展，如果有，则要用剩余高地址线通过译码统一进行片选选择，而 8031 因无片内 ROM，还要同时进行外部程序存储器扩展。

7.4 并行 I/O 口的应用

在计算机应用系统中，因系统扩展外部存储器而占用 P2 和 P0 口，而 P3 口又被作为

第二功能使用时，留给用户的只有 P1 口，这时不可避免地要进行 I/O 口的扩展，以便有效地与外部设备相连接。

由于 MCS－51 系列单片机的外部 RAM 和 I/O 口是统一编址的，因此用户可以把单片机的外部 64 KB RAM 空间的一部分作为扩展 I/O 的地址空间。这样，单片机就可以像访问外部 RAM 存储器那样访问外部接口芯片，对其端口进行读/写操作。

最常用的 I/O 扩展芯片有 8255(3×8 并行口)、8243(4×4 并行口)等专用接口芯片，8155(2 个 8 位并行口，1 个 6 位并行口，256B 静态 RAM，1 个 14 位定时器/计数器)、8755(2 个 8 位并行口，2 K×8 bit EPROM)等复合接口芯片以及如 74LS373、74LS165 等 TTL 电路芯片。

7.4.1 I/O 口的简单扩展

图 7－13 所示是一个采用缓冲器 74LS244 作为扩展输入、锁存器 74LS273 作为扩展输出的简单 I/O 扩展应用。

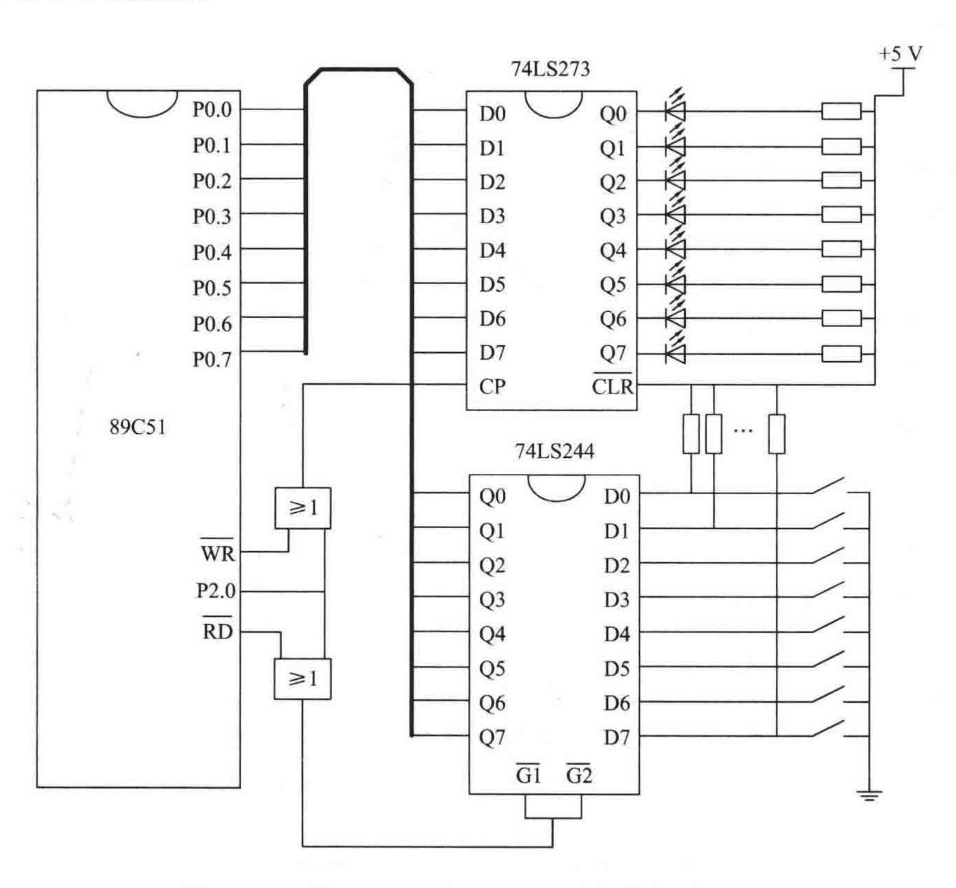


图 7－13 用 74LS273 和 74LS244 进行简单 I/O 扩展

其中，74LS244 是 8 路缓冲线驱动器(三态输出)。它将 8 个三态线驱动器分成两组，分别由低电平有效的$\overline{G1}$、$\overline{G2}$控制，当两者中出现高电平时，有关输出为三态。74LS273 是 8D 触发器，其中$\overline{CLR}$为低电平有效的清除端，当$\overline{CLR}$＝0 时，输出全为 0 且与其他输入端无关。

P0 口作为双向 8 位数据线，既能够从 74LS244 输入数据，又能够从 74LS273 输出数据。

输入控制信号由 P2.0 和$\overline{RD}$相“或”后形成。当二者都为 0 时，74LS244 的控制端$\overline{G1}$、$\overline{G2}$有效，选通 74LS244，则外部的输入信息(从 D0～D7 引脚输入)就能从 Q0～Q7 输出。

输出控制信号由 P2.0 和$\overline{WR}$相“或”后形成。当二者都为 0 时，74LS273 的控制端 CP 有效，选通 74LS273，则从 D0～D7 引脚输入的数据就能锁存到 74LS273 的输出端。

【例 7-3】 如图 7-13 所示，编写一段程序，功能是按下任意键，使对应的 LED 发光。

分析：当 74LS244 选通时，与 74LS244 相连的按控信息将会从 D0～D7 引脚输入，并从 Q0～Q7 输出，再输入到 P0 口的数据总线上。

当 74LS273 选通时，从 P0 口输出的数据将被锁存到 74LS273 的输出端，控制 Q0～Q7 引脚上接的发光二极管 LED。

因为 74LS244 和 74LS273 都是在 P2.0 为 0 时被选通的，所以二者的口地址都可以是 FEFFH(不是唯一的，只要保证 P2.0=0)。但由于分别由$\overline{RD}$和$\overline{WR}$控制，这两个信号都是在执行 MOVX 指令时产生的，因此，两者不可能同时有效，所以在逻辑上，输入和输出不会产生冲突。

参考程序：

```
        ORG     0000
        LJMP    START
        ORG     1000H
START:  MOV     DPTR, #0FEFFH
        MOVX    A, @DPTR            ; 通过 74LS244 读开关状态
        MOVX    @DPTR, A            ; 通过 74LS273 控制灯亮灭
        LJMP    START
        END
```

7.4.2 LED 数码管显示接口

显示器是最常用的输出设备，在单片机应用系统中，常用发光二极管和数码管作为显示器显示信息。由于数码管结构简单、价格便宜、接口容易，在单片机系统中被大量使用。

1. LED 数码管显示器的结构

LED 数码管显示器的结构如图 7-14 所示。LED 数码管显示器内部由 8 个发光二极管组成。其中 7 个长条形的发光二极管排列成“日”字形，另一个圆点形状的发光二极管在显示器的右下角，用于显示小数点。当二极管导通时，相应的笔画段发亮。因此，只要分别控制各笔画段的发光二极管，使其中的某些发亮，就可以显示各种不同的字符。

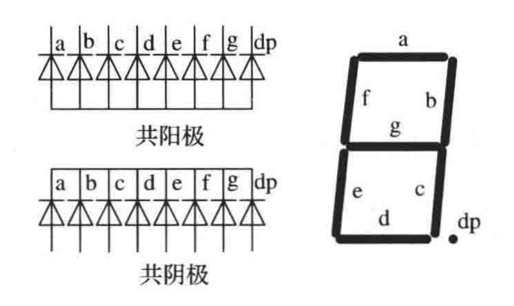


图 7-14 LED 数码管显示器的结构

LED 数码管显示器的内部结构有两种：

(1) 共阳极结构：8 个发光二极管的阳极全部连接在一起组成公共端，8 个发光二极管的阴极单独引出，当公共端接高电平时，只要相应的阴极出现低电平，对应的发光二极管就会发亮。

(2) 共阴极结构：8 个发光二极管的阴极全部连接在一起组成公共端，8 个发光二极管的阳极单独引出，当公共端接低电平时，只要相应的阳极出现高电平，对应的发光二极管就会发亮。

无论是共阴极还是共阳极结构的 LED 显示器，它们排列成“日”字形的各笔画段的安排顺序都是相同的，如图 7－14 中的“a、b、c、d、e、f、g、dp”。

2. LED 数码管的驱动方法

在单片机应用系统中，LED 数码管显示器的显示方法有两种：静态显示法和动态扫描显示法。

1) 静态显示法

图 7－15 所示为一个 4 位的静态显示器电路。

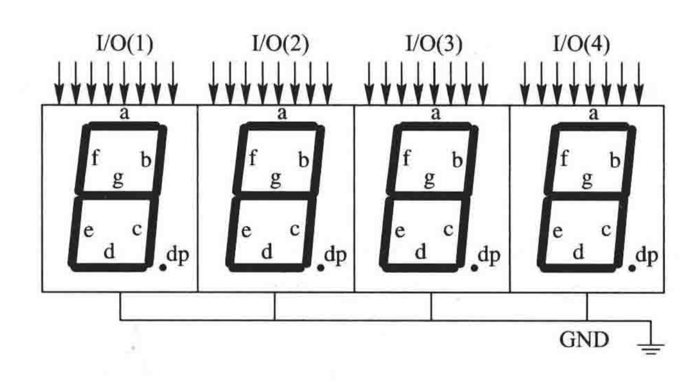


图 7－15　4 位 LED 静态显示电路

所谓静态显示，就是当显示器显示某一个字符时，相应的发光二极管恒定地导通或截止。例如，七段显示器的 a、b、c、d、e、f 导通，g 截止，则显示 0。这种显示方式，每一位都需要有一个 8 位输出口控制，所以占用硬件多，一般用于显示器位数较少(很少)的场合。

在静态显示电路中，由于 I/O 只要有段码输出，相应字符就会显示出来，并保持不变，直到 I/O 输出新的段码，因此，为防止烧坏 LED，一般要在 LED 的 8 个发光二极管的各引出端接限流电阻，阻值大小根据额定导通电流来确定。

当位数较多时，用静态显示所需的 I/O 口太多，一般采用动态显示方法。

用 74LS164 扩展两个 8 位静态数码显示电路设计如图 7－16 所示。单片机的串行口工作在方式 0，即移位寄存器方式，74LS164 是 8 位串入/并出的移位寄存器。显示数据时，串行数据由单片机的 P3.0(RxD)送出，同步移位时钟脉冲由 P3.1(TxD)送出。在移位时钟脉冲的作用下，串行口发送缓冲器 SBUF 中的数据按先后顺序逐位移入 74LS164 移位寄存器中，于是两片 74LS164 的并行输出口将并行输出移入的数据，分别驱动两个 LED 数码管显示数据。

【例 7－4】 试将 RAM 缓冲区 40H、41H 中的单 BCD 码通过 74LS164 并行输出并显示到两位 LED 数码管，如图 7－16 所示。假设 LED 数码管是共阳的。

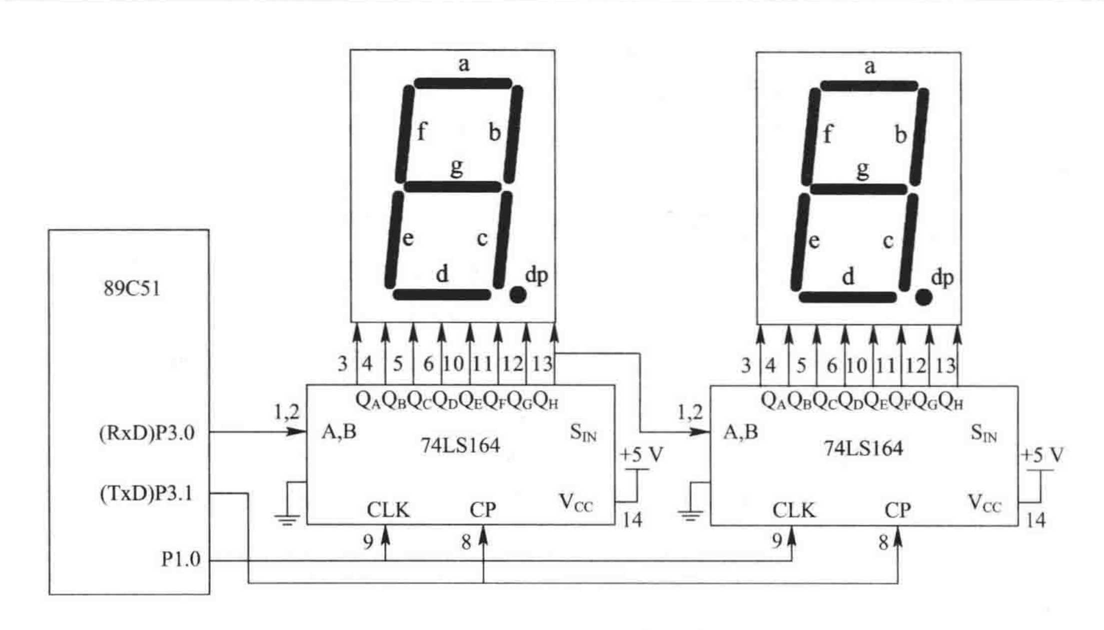


图 7-16　74LS164 驱动的静态数码显示电路

分析：40H 和 41H 中的单 BCD 码是“0～9”，需读取字形码，再放入到 40H 和 41H；设置串行口工作于方式 0，然后将 P1.0 置“1”，选通 74LS164，最后调用一个串行输出 16 位的子程序。

参考程序：

```
        ORG    0000H          ；主程序，主要是读取字形码
        MOV    R7，#02H       ；设置显示的位数
        MOV    R0，#40H       ；设置地址指针
        MOV    A，@R0         ；将 BCD 码读入 A 中
LOOP：  MOV    DPTR，#TAB     ；取字形码的基址
        MOVC   A，@A+DPTR     ；读字形码
        MOV    @R0，A         ；存回缓冲区
        INC    R0             ；取下一个数
        DJNZ   R7，LOOP       ；循环
        MOV    R7，#02H       ；设置要发送的字节数
        MOV    R0，#40H       ；设置地址指针
        SETB   P1.0           ；选通 74LS164
        LCALL  DISP           ；调用显示子程序，显示两位 BCD 码
        CLR    P1.0           ；关闭 74LS164
        LJMP   $
DISP：  NOP                   ；显示两位 BCD 码的子程序
        MOV    SCON，#00H     ；设置串行口工作于方式 0
SEND：  MOV    A，@R0         ；取发送数据
        MOV    SBUF，A        ；启动发送数据
WAIT：  JNB    TI，WAIT       ；一帧数据未发送完，循环等待
        CLR    TI             ；清除发送中断标志 TI
```

```
        INC     R0              ；取下一个数
        DJNZ    R7, SEND        ；循环
        RET                     ；子程序返回
TAB:    DB      0C0H, 0F9H, 0A4H, 0B0H, 99H, 92H, 82H, 0F8H, 80H, 90H
                                ；0～9 对应的字形码
        END
```

2）动态扫描显示法

在多位 LED 显示时，为了简化电路，降低成本，一般采用动态扫描显示方式。动态扫描显示是单片机应用系统中最常用的显示方式之一。图 7－17 就是一个 4 位 LED 动态显示电路。

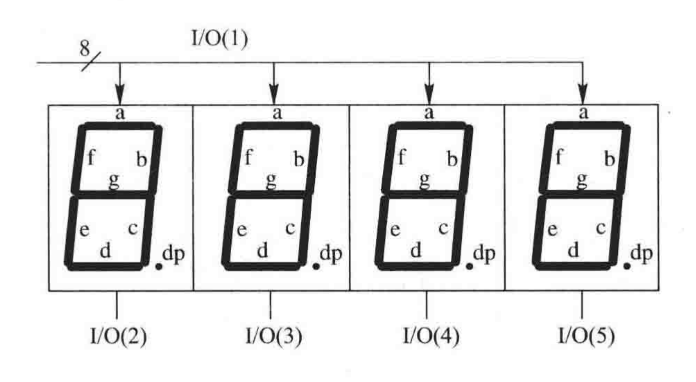


图 7－17　4 位 LED 动态显示电路

所谓动态显示，就是一位一位地轮流点亮各位显示器(扫描)，对于每一位显示器来说，每隔一段时间点亮一次，但由于人眼存在视觉暂留效应，加上发光二极管的余辉效应，只要扫描的速度足够快，每位显示的间隔时间足够短，就可以给人同时显示的感觉，而不会有闪烁感。调整电流和显示时间间隔，可实现亮度较高且稳定的显示。

在动态显示电路中，所有 LED 显示器的 8 个笔画段的各段同名端互相连接起来，并把它接到输出 I/O(1)口上(称为数据口，8 位)，这样 I/O 上输出的字形码会同时到达每个 LED 的“dp、g、f、e、d、c、b、a”。为了防止各显示器显示同样的数字，各个显示器应该轮流显示，在某一时刻只能是其中的一个数码管点亮，因此每个数码管的 COM 端还要受到另一信号的控制，方法是将 COM 端接到另外一个 I/O 输出口(称为扫描口，位数与显示器位数相等)上，某一个时刻只让其中的一个 COM 出现低电平或高电平。

在 MCS－51 单片机应用系统中，若无需扩展外部存储器等部件，可直接用单片机自带的 I/O 口来构建数码管动态显示系统。若外部扩展占用了 I/O 资源，可以采用 8155、8255 等芯片扩展 I/O 口，构建数码管动态显示系统。

【例 7－5】 图 7－18 为利用单片机 P2 口和 P1 口构成的动态显示的电路，共有 6 个共阳极 LED 数码管显示器，P2 口为字段口，输出字形码，P2.0～P2.7 分别与“a、b、c、d、e、f、g、dp”对应相连，P1 口为字位口，输出位码。编写程序，使图 7－18 的动态扫描显示电路从左到右显示 1、2、3、4、5、6 共六个字符。设晶振频率为 12 MHz。

分析： 在第一时刻从 P2 口输出 1 的字形码，从 P1 口输出 00100000B(20H)，使最左边的 LED 点亮；延时一段时间后，从 P2 口输出 2 的字形码，从 P1 口输出 00010000B

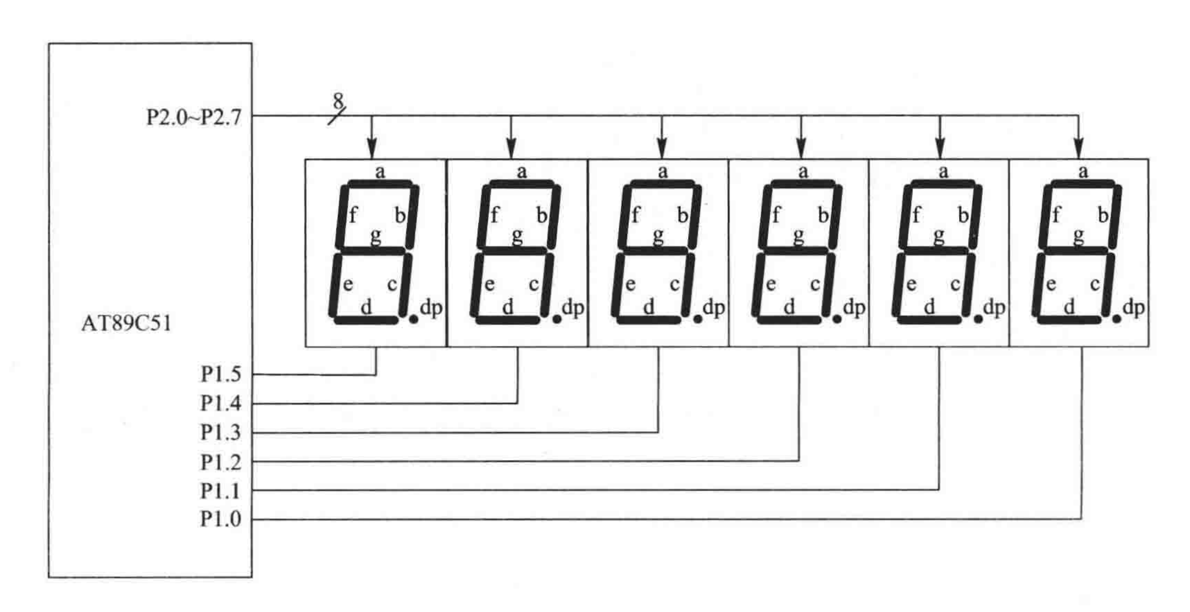


图 7-18 6 位 LED 动态显示电路

(10H)，使左边第二个 LED 点亮；依次循环，最后从 P2 口输出 6 的字形码，从 P1 口输出 0000001B(01H)，使最右边的 LED 点亮；再回过头从左边第一个开始。

获取字形码采用查表方式，将字形码以表格的形式按顺序存储到 ROM 中，将表格首地址取到 DPTR 中，将要显示的数字送给累加器 A，执行"MOVC A，@A+DPTR"指令获取字形码。

参考程序 1：

```
        ORG     0000H
START:  MOV     R0，#06H          ；程序循环计数器，六个字符一循环
        MOV     R1，#00H          ；字形码的偏移量，首先对应字符"0"
        MOV     R2，#20H          ；位码，首先 P1.5=1，然后依次类推
DISP:   MOV     DPTR，#TAB        ；取字形码首地址
        MOV     A，R1
        MOVC    A，@A+DPTR        ；读取字形码
        INC     R1                ；指向下一个字形码
        MOV     P2，A             ；字形码送 P2 口
        MOV     A，R2
        MOV     P1，A             ；位码送 P1 口
        RR      A                 ；右移，指向下一个位码
        MOV     R2，A             ；位码移位，为显示下一个做准备
        LCALL   DELAY             ；调用延时子程序
        DJNZ    R0，DISP          ；不等于 0，则继续循环
        LJMP    START             ；等于 0，则重新赋初值，再继续循环
TAB:    DB      0F9H，0A4H，0B0H，99H，92H，82H    ；1~6 对应的字形码
DELAY:  MOV     R7，#40           ；延时子程序，延时时间约为 10 ms
DEL1:   MOV     R6，#123
        NOP
```

```
DJNZ    R6, $
DJNZ    R7, DEL1
RET
END
```

对于动态扫描显示而言，由于各数码管大部分时间不亮，只有一小部分时间亮，因此在设计实际的硬件电路时，并不一定需要加限流电阻，可以将单片机输出段码的I/O直接接到LED的8个发光二极管的各引出端。

采用动态显示方式比较节省I/O口，硬件电路也较简单，但其亮度不如静态显示方式。而且在显示位数较多时，CPU要依次扫描，占用CPU较多的时间。我们已经知道，一旦程序使用软件延时，在CPU执行延时程序的时候，就不能做其他的事，这样势必会降低CPU的效率。在实际应用中，当然不可能只显示几个数字，还要做其他的事。这时，我们可以借助定时器。定时时间一到，产生中断，更换一个数码管点亮，然后立刻返回；此次点亮的数码管会一直亮到下一次定时时间到。这段时间内不执行延时程序，可以留给主程序做其他的事。到下一次定时时间到，则点亮下一个数码管。

参考程序2(采用定时器中断方法)：

```
        ORG     0000H
        LJMP    START
        ORG     000BH           ；定时器T0入口地址
        LJMP    TIME
START:  MOV     R0, #06H        ；程序循环计数器，六个字符一循环
        MOV     R1, #00H        ；存字形码的偏移量，首先对应字符“0”
        MOV     R2, #20H        ；位码，首先P1.5=1，然后依次类推
        MOV     TMOD, #01H      ；T0设置成工作方式1，定时模式
        MOV     TL0, #3CH
        MOV     TH0, #0F6H      ；T0赋初值，定时2.5 ms
        SETB    ET0             ；T0开中断
        SETB    EA              ；开总中断
        SETB    TR0             ；启动T0
        LJMP    $
TIME:   MOV     DPTR, #TAB      ；T0中断服务程序，取字形码首地址
        MOV     A, R1
        MOVC    A, @A+DPTR      ；读取字形码
        INC     R1              ；指向下一个字形码
        MOV     P2, A           ；字形码送P2口
        MOV     A, R2           ；取位码
        MOV     P1, A           ；位码送P1口
        RR      A               ；右移，指向下一个位码
        MOV     R2, A           ；保存位码，为显示下一个做准备
        DJNZ    R0, T10         ；不等于0，则继续循环
        MOV     R0, #06H        ；程序循环计数器重新赋值
```

```
        MOV     R1, #00H            ;字形码的偏移量重新赋值
        MOV     R2, #20H            ;位码重新赋值
T10:    MOV     TL0, #0B0H
        MOV     TH0, #3CH           ;T0 重新赋初值
        RETI                        ;中断返回
TAB:    DB      0F9H, 0A4H, 0B0H, 99H, 92H, 82H    ;1~6 对应的字形码
        END
```

7.4.3 键盘接口

在单片机应用系统中，为了控制系统的工作状态，以及向系统输入数据，应用系统应设有按键或键盘。键盘按接口形式可以分为独立连接式和行列(矩阵)式两类，按结构形式可分为编码键盘和非编码键盘。

编码键盘除了按键以外，还包括产生编码的硬件芯片(如：8279)；非编码键盘靠软件来识别键盘上的闭合键，由此得出键值，在单片机应用系统中被普遍采用。

1. 非编码键盘的设计原理

非编码键盘的设计必须解决以下问题：

(1) 判定是否有键按下。

(2) 若有键按下，判定是哪个键按下，确定被按键的“键值”。

(3) 除抖动。按键从最初的按下到接触稳定要经过数毫秒的抖动时间，键松开时也有同样的问题，如图 7-19 所示。抖动会引起一次按键多次读数，实际使用时必须避免。可以用硬件或软件方法来消除抖动。通常，键数较少时，采用硬件消除抖动，如用 RS 触发器消除抖动(见图 7-20)；键数多时，常用软件消除抖动。当检出键按下后执行一个延时子程序产生数毫秒的延时，使前沿抖动消失后再检验键的闭合；当发现键松开后，也要经数毫秒的延时，待抖动消失后再检验下一次键的闭合。

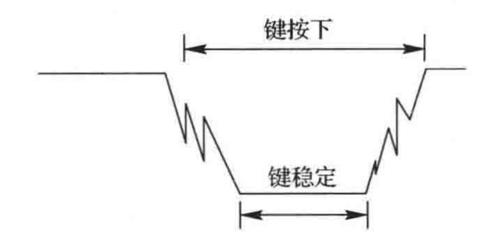




图 7-19 键闭合及断开时的电压抖动

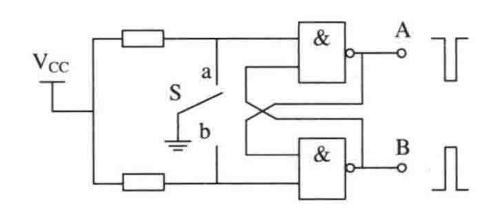


图 7-20 由 RS 触发器构成的去抖电路

(4) 准确得出按键值(或键号)，以满足跳转指令要求。键盘上的每个键都应有一个键值，CPU 将根据它来执行相应的功能程序。为了方便地使用散转指令，在许多场合下，常采用依次排列键值的方法，这时的键值与键号一致。比如对图 7-23 中的矩阵键盘，就可以将 S1～S16 号键的键值直接定为 00H～0FH，这样，扫描后得到的键值(存放在 A 中)可以直接用于散转指令。指令系统中的散转指令“JMP @A+DPTR”，就是专门为了配合键输入信息而设置的。

(5) 同一按键长时间持续按下。在一般情况下，无论一次按键的时间有多长，系统仅

执行一次按键功能程序，但是很多智能仪表的数据修改键恰好充分利用这一特点，来增加数据的修改速度。

（6）处理同时按键。由于硬件条件限制，系统可以用多个按键的组合键实现不同功能，若没有组合键功能，也可采用对按键位的查询顺序来确定哪个按键有效。

2. 单片机对非编码键盘的控制方式

单片机获取按键状态有三种方式：程序查询方式、定时扫描方式和中断扫描方式。

1）程序查询方式

采用程序查询方式将使 CPU 时刻处于键盘检测状态，不能做其他事。对于单片机应用系统，键盘处理往往只是 CPU 工作的一部分，所以此种方式效率低下。

2）定时扫描方式

单片机对键盘的扫描也可以采用定时扫描方式，即每隔一定的时间对键盘扫描一次。在这种扫描方式中，通常利用单片机的定时器，产生 10 ms 的定时中断，CPU 响应定时器溢出中断请求，调用键盘扫描子程序，以响应键盘输入请求。键盘扫描子程序流程图如图 7－21 所示，其中 KM 为除抖动标志位。需要说明的是，键按下或键抬起都会执行按键处理程序，所以我们还需在按键处理程序前识别按键是按下还是抬起，以便执行不同的程序来分别处理。

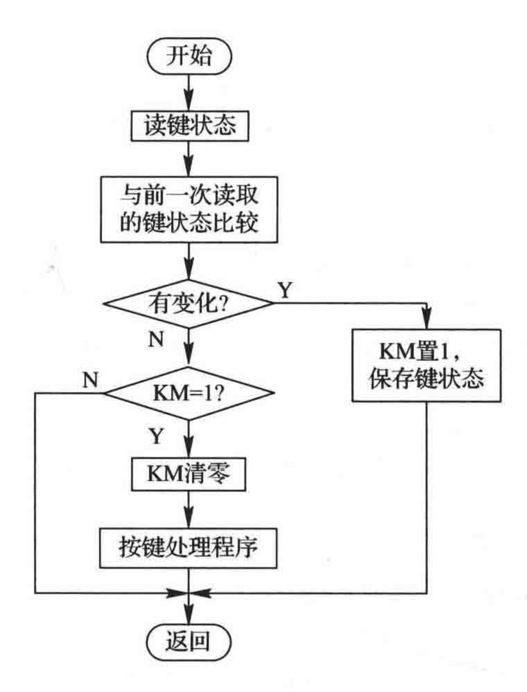


图 7－21　键盘扫描子程序流程图

3）中断扫描方式

在键盘定时扫描方式中，CPU 总要定时扫描，若此时定时中断程序仅仅是进行键盘扫描，则在大多数情况下，CPU 对键盘是空扫描。为提高 CPU 的效率，可以采用中断扫描方式。当键盘有键闭合时，产生中断请求，CPU 响应中断，执行服务程序，判断键号，做相应处理。

3. 独立式按键接口电路

独立式按键是指直接用 I/O 口线构成的单个按键电路。每个独立式按键单独占用一

位 I/O 口线。独立式按键电路如图 7－22 所示。

独立式按键电路配置灵活，软件结构简单。但由于每个按键必须单独占用一根 I/O 口线，在按键数量较多时，I/O 口线浪费较大。如果应用系统中的键较少，就可采用独立式的键盘接口电路。

【例 7－6】 根据图 7－22 所示的独立式按键电路，编写程序查询方式的键盘处理子程序。

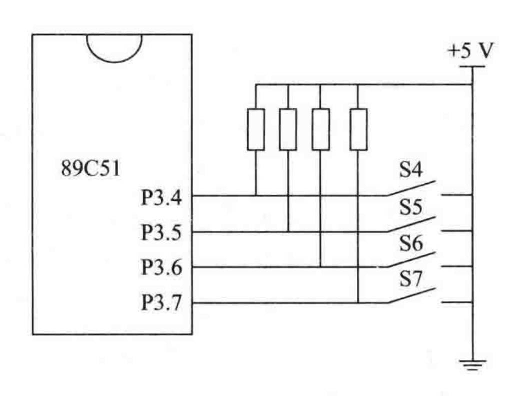


图 7－22　独立式按键电路

分析 1： 假设 I/O 口是 P3 口，当某一按键 Sn(n＝4～7)闭合时，P3.n 输入为低电平；释放时，P3.n 输入为高电平。

在程序中 S4～S7 为每个按键功能程序入口地址标号，PROM4～PROM7 为每个按键的功能程序。

参考程序 1：

```
RDKEY：  MOV    P3，＃0FFH     ；准双向口作为输入口，则需先向对应口输出"1"
         MOV    A，P3          ；P3 口输入
         JNB    ACC.4，S4      ；以下均是判断去向
         JNB    ACC.5，S5
         JNB    ACC.6，S6
         JNB    ACC.7，S7
DONE：   RET
S4：     LJMP   PROM4
S5：     LJMP   PROM5
S6：     LJMP   PROM6
K7：     LJMP   PROM7          ；转对应服务程序
PROM4：  ⋮
         LJMP   DONE
PROM5：  ⋮
         LJMP   DONE
PROM6：  ⋮
         LJMP   DONE
PROM7：  ⋮
         LJMP   DONE           ；程序执行后返回
```

分析 2： 可以采用散转指令来编写程序。S4 闭合时，P3 输入 1110××××；S5 闭合时，P3 输入 1101××××；S6 闭合时，P3 输入 1011××××；S7 闭合时，P3 输入 0111××××。将对应值转换成 00000000、00000001、00000010、00000011，即 0～3；采用带进位位右移的方法，通过进位位 Cy=0 来计算移位次数，即是需要的键值；然后转换值乘 3 得到偏移量。需要注意的是：低 4 位没用，应先右移 4 次。

参考程序 2：

```
RDKEY: MOV   P1, #0FFH     ；准双向口作为输入口，需先向对应口输出“1”
       MOV   A, P3         ；P3 口输入
       MOV   B, #00H       ；设置一个工作计数器
       RRC   A
       RRC   A
       RRC   A
       RRC   A
RD10:  RRC   A             ；带进位位右移
       JNC   RD20
       INC   B             ；工作计数器+1
       LJMP  RD10
RD20:  MOV   A, #03H
       MUL   AB
       MOV   DPTR, #TAB
       JMP   @A+DPTR
DONE:  RET
TAB:   LJMP  PROM4         ；A=0，执行 PROM4
       LJMP  PROM5         ；A=3，执行 PROM5
       LJMP  PROM6         ；A=6，执行 PROM6
       LJMP  PROM7         ；A=9，执行 PROM7
PROM4: ⋮
       LJMP  DONE
PROM5: ⋮
       LJMP  DONE
PROM6: ⋮
       LJMP  DONE
PROM7: ⋮
       LJMP  DONE          ；程序执行后返回
```

4. 矩阵式键盘

1）矩阵式键盘的结构

在键盘中按键数量较多时，为了减少 I/O 口的占用，通常将按键排列成矩阵形式，如图 7-23 所示。在矩阵式键盘中，每条水平线和垂直线在交叉处通过一个按键加以连接。这样，一个 8 位并行端口(如 P3 口)最多可以构成 4×4=16 个按键，比直接将端口线用于键盘多出了一倍，而且线数越多，区别越明显。比如，再多加一条线就可以构成 20 键的键盘，而直接用端口线则只能多出一键(9 键)。

2）矩阵式键盘的工作原理及按键的识别

矩阵式键盘的识别比独立式要复杂一些。在图 7－23 中，行线和列线所接的 I/O 口分别作为输入端和输出端。这样，当按键没有按下时，所有的输入端都是高电平，代表无键按下。若某输出是低电平，一旦有键按下，则输入线就会被拉低，这样，通过读入输入端的状态即可知是否有键按下。识别按键的方法很多，其中最常用的是扫描法和反转法。

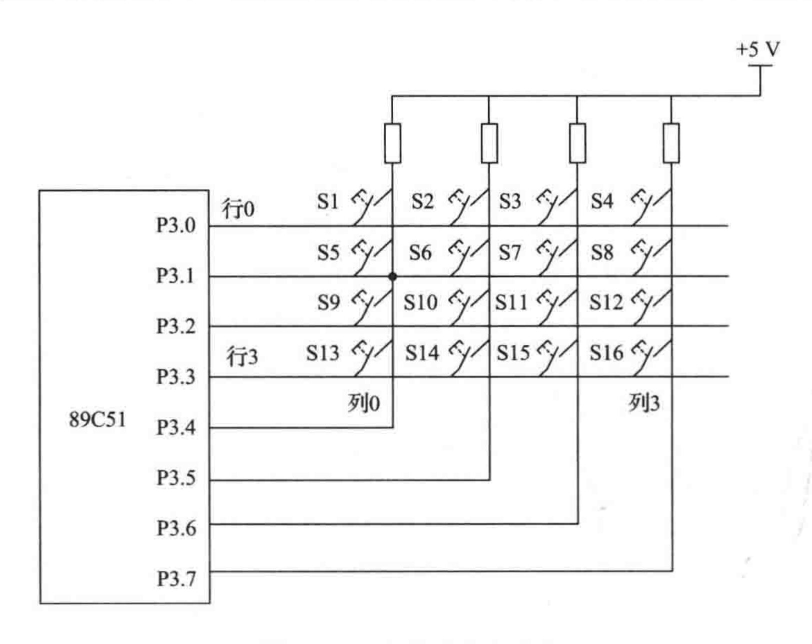


图 7－23　矩阵式按键电路

（1）扫描法。

在图 7－23 中，列线通过电阻接正电源，并将行线所接的 I/O 口作为输出端，列线所接的 I/O 口作为输入端。这样，任何按键都没有按下时，所有的列输入端都是高电平。当所有的行线都处于高电平时，按键按下与否不会影响列线电平的变化，因此，必须使行线处于低电平，只有这样，当有键按下时，该键所在的列电平才会由高电平变为低电平，CPU 根据列线电平的变化，便能判断相应的列是否有键按下。以图 7－23 中 4×4 键盘的 S5 号键的识别为例来说明，当 S5 键按下后，与此键相连的行线和列线导通，列 0 肯定是低电平。然而列 0 为低电平时能否肯定仅是 S5 键按下呢？当然不行，因为 S1、S9、S13 号键按下也同样可能会使列 0 为低电平。

为进一步确定具体键，不能使所有的行线在同一时刻都处于低电平。当某一行线输出是低电平时，比如行 1 处于低电平且行 0、2、3 处于高电平时，我们扫描的是 S5、S6、S7、S8 四个键是否按下，如 S5 键按下，则列 0 输入为低，如 S8 键按下，则列 3 输入为低。这样，通过读入列输入线（列 0）的状态并结合行输出线的状态（行 1）就可得知 S5 键是否按下。以此类推，其他键是否按下，也可以通过这个方法确定。

所以，在某一时刻只能使 1 条行线处于低电平，其余行线都处于高电平；另一时刻，让下一行处于低电平，依次循环。这种工作方式称为键盘扫描。假设我们只考虑按键按下时有效（有些场合按键抬起时也要求系统能完成一定功能），那么一个完整的键盘扫描程序应包括以下内容：

① 检测当前是否有键按下。方法是输出全 0 信号到所有的行线上，然后读所有列线的

状态。当所有列线不全为1时，表明有键闭合；否则表明无键闭合，继续等待。

② 当有键闭合时，可以采用硬件措施或调用软件延时程序消除抖动。

③ 在确认键已稳定闭合后，需要进一步判断是哪一个键闭合。方法是对键盘进行扫描，就是依次给每一条行线送出0信号，其余各行线均为1，并相继检测每一次扫描时所对应的列状态。若各列全为1，则表示为0的这一行上没有键闭合；否则为0的这一行上有键闭合，且闭合键所在的列就是列状态为0的列。

④ 判断闭合键是否释放，如没释放，则继续等待。

⑤ 用查表法或计算法得到键值，然后转向相应的处理程序。

键盘扫描也可以采用扫描列线判断行线的方法。

【例7-7】 如图7-23所示矩阵式键盘，编写键值获取子程序。规定：S1～S16键对应的键值为0～15，如果有键按下，键值存入A，以便用散转指令转移到不同的处理程序上去。如果没有键按下，则A中的值为FFH。

分析(扫描法)：参考程序是一个子程序，入口名称为KEYP。出口参数为A(即键值)。首先判断是否有键按下，用R3做行扫描控制，初始值为00000001B；然后依次左移，共4次，从上到下分别扫描4条行线，R4为行号记录寄存器，取值0、4、8、12，以便于直接计算键值。假设x为行位置，y为列位置，则键值为4x+y。由于x、y的取值均为0～3，故最终的键值为0～15。

参考程序(扫描法)：

```
KEYP:   MOV     P3, #0F0H           ; P3.4～P3.7对应列线，置成输入状态
        MOV     A, P3               ; 读P3口的值
        ANL     A, #0F0H
        CJNE    A, #0F0H, PX00      ; 不全为1，有键按下
        LJMP    DONE                ; 无键按下，返回
PX00:   LCALL   DELAY               ; 延时子程序，在本例中省略(可以参考例7-5)
                                    ; 以下4句用于判断键盘抖动
        MOV     A, P3               ; 再次从P3口读入
        ANL     A, #0F0H
        CJNE    A, #0F0H, PX11      ; 确认有键按下
        LJMP    DONE                ; 抖动，返回
PX11:   MOV     R2, #04H            ; 扫描次数
        MOV     R3, #00000001B      ; 行扫描记录位(低4位)
        MOV     R4, #00H            ; 行号记录寄存器，初值=0
SCA:    MOV     A, R3
        CPL     A                   ; 取反，将扫描行清"0"，其他行置"1"
        MOV     P3, A               ; 输出行线
        MOV     A, P3               ; 读列线
        ANL     A, #0F0H
        CJNE    A, #0F0H, FKN1      ; 有键按下
        MOV     A, R3               ; 修改行扫描线状态
        RL      A
```

```
        MOV     R3, A
        MOV     A, R4           ；行号自增4
        ADD     A, #04
        MOV     R4, A
        DJNZ    R2, SCA         ；判断4次扫描是否完成
        LJMP    DONE            ；无键按下，返回
FKN1:   MOV     R2, #04H        ；有键按下，循环4次，获得行号
        SWAP    A               ；把行线的值交换到低4位
FKN2:   RRC     A               ；A中是行线位置，R4中为0、4、8、12中的一个
        JNC     FKN3
        INC     R4              ；获取键值
        DJNZ    R2, FKN2
FKN3:   MOV     A, R4           ；有键按下，键值存入A
        LJMP    DONE1
DONE:   MOV     A, #0FFH        ；没有键按下，A←FFH
DONE1:  RET
```

【例7-8】 设图7-23中16个按键对应的处理程序为WORK0～WORK15，利用例7-7获得的键值，应用散转指令跳转到按键对应的处理程序上去。

分析： 将获得的键值(在A中)乘以3，得到偏移量，直接使用散转指令"JMP @A+DPTR"，跳到相应位置。

参考程序：

```
START: LCALL    KEYP            ；调用子程序，读键值
       CJNE     A, #0FFH, TRUE  ；比较，有效值(0～15)则转键盘处理程序
       LJMP     START           ；否则，继续判断键盘，获取键值
TRUE:  MOV      B, #03
       MUL      AB              ；键号乘以3
       MOV      DPTR, #KNOW
       JMP      @A+DPTR
KNOW:  LJMP     WORK0           ；跳转到S1键对应处理程序
       LJMP     WORK1
                ⋮
       LJMP     WORK15          ；跳转到S16键对应处理程序
WORK0:          …               ；S1键对应处理程序
WORK1:          …
                ⋮
WORK15:         …               ；S16键对应处理程序
```

(2) 反转法。

扫描法要逐行(列)扫描查询，当所按下的键在最后行(列)时，要经过多次扫描才能获得键值或键号。而采用反转法，只要经过两个步骤即可获得键值。

反转法原理如图7-24所示，用一个8位I/O口构成4×4键盘，I/O口的排列规律与

图 7－23 相同。假设图中所画为所按下的键，则实现例 7－7 键盘功能的反转法原理如下：

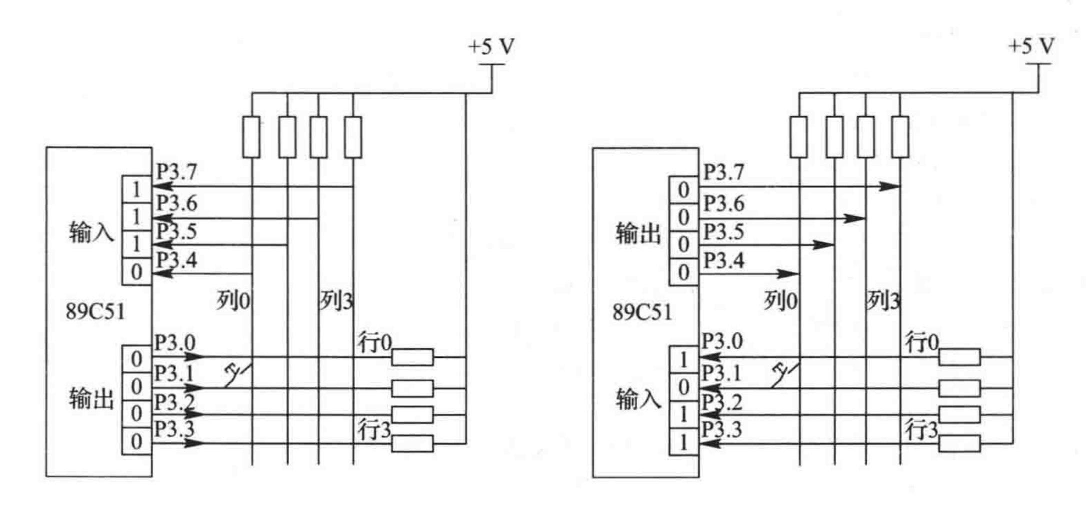


图 7－24 反转法原理图

第一步，行输出线(P3.3～P3.0)输出 0000，若有键按下，则列输入线(P3.7～P3.4)为 0 的位对应的是被按下键的列位置。如按下的是图中所示的键，则输入为 1110，按键位于第 0 列，并将这个输入数据暂存入 B(或某存储单元)中。

第二步，列输出线(P3.7～P3.4)输出 0000，若有键按下，则行输入线(P3.3～P3.0)为 0 的位对应的是被按下键的行位置。如按下的是图中所示的键，则输入为 1101，按键位于第 1 行。

综合这两步，就知道按键所在的位置，图示按键在第 1 行第 0 列。

下面需要计算行号和列号：通过移位法，将行数据右移，遇到“0”结束，同时得出行号并乘以 4；将列数据交换到低 4 位，列数据右移，得到列号，然后相加，即是键值。

参考程序(反转法)：

```
KEYP:   MOV    P3，#0F0H           ；行线输出“0”，同时列线输出“1”，为列线读入做准备
        MOV    A，P3               ；读入列线数据
        ANL    A，#0F0H            ；保留列线数据，屏蔽其他位
        CJNE   A，#0F0H，PX00      ；不全为 1，有键按下
        LJMP   DONE                ；无键按下，返回
PX00:   LCALL  DELAY               ；延时，以下 4 句用于判断键盘抖动
        MOV    A，P3               ；再次从 P3 口读入
        ANL    A，#0F0H            ；保留列线数据，屏蔽其他位
        CJNE   A，#0F0H，PX11      ；确认有键按下
        LJMP   DONE                ；抖动，返回
PX11:   MOV    B，A                ；存列线数据
        MOV    P3，#0FH            ；列线输出“0”，同时行线输出“1”，为行线读入做准备
        MOV    A，P3               ；读入行线数据
        MOV    R4，#00H
PX22:   RRC    A                   ；右移，计算行
        JNC    PX33                ；遇到“0”结束循环
```

```
        INC     R4          ；行号自增 4
        INC     R4
        INC     R4
        INC     R4
        LJMP    PX22
PX33：  MOV     A，B        ；列线送 A
        SWAP    A           ；把列线数据交换到低 4 位
PX44：  RRC     A           ；右移，计算列
        JNC     PX55        ；遇到“0”结束循环
        INC     R4          ；列号自增 1
        LJMP    PX44
PX55：  MOV     A，R4       ；有键按下，键值存入 A
        LJMP    DONE1
DONE：  MOV     A，#0FFH    ；没有键按下，A←FFH
DONE1： RET
```

7.5 A/D、D/A 转换器及应用

在单片机应用系统中，常常需要把检测到的连续变化的模拟信号(如流量、温度、压力、液位等)转换成数字信号，才能送入到单片机中进行各种处理。单片机对这些从外部获取的各种数据进行处理后，再将这些数字量转换成模拟量，并输出到外部对被控对象进行控制。将模拟信号转换成数字信号的过程称为 A/D 转换，将数字信号转换成模拟信号的过程称为 D/A 转换。A/D 和 D/A 这些器件与单片机之间进行信息交换主要采用并行和串行两种方式，本节主要介绍并行数据交换方式。

7.5.1 A/D 转换器

随着单片机技术的不断发展，许多新一代的单片机已经在片内集成了多路 A/D 转换通道和 PWM 输出，这大大简化了电路和编程工作，但这类 CPU 芯片大多价格较贵。本节主要介绍 CPU 芯片内无 A/D 转换电路的 MCS-51 单片机与 A/D 芯片的接口技术。

1. A/D 转换器的主要技术指标

1) 量化误差与分辨率

分辨率反映 A/D 转换器所能分辨的被测量的最小值，通常以输出二进制位数或 BCD 码值数表示。位数越多，量化分层越细，分辨率就越高。例如，一个 A/D 转换器为 8 位，则其分辨率为满刻度电压的 $1/2^8$，该 A/D 转换器能分辨的最小电压值为 $5/2^8=20$ mV；若 A/D 转换器为 10 位，则该 A/D 转换器能分辨的最小电压值为 $5/2^{10}=4.9$ mV。

为将模拟信号转换为数字量，在 A/D 转换过程中，还必须将采样/保持电路的输出电压按某种近似方式归化到相应的离散电平上，这一转化过程称为数值量化，简称量化。量化过程中所取最小数量单位称为量化单位，用 Δ 表示，记为 1LSB(Least Significant Bit)。在量化过程中，由于采样电压不一定能被 Δ 整除，所以量化前后不可避免地存在误差，此

误差称为量化误差。对于四舍五入量化方式，最大量化误差理论上为±1/2 LSB。

2）转换精度

A/D转换器的转换精度反映了一个实际A/D转换器在量化值上与一个理想A/D转换器进行A/D转换的差值，可表示成相对误差和绝对误差。常用数字量的位数作为度量绝对精度的单位，如精度为±1/2 LSB，而用百分比来表示满量程时的相对误差，如±0.05%。

注意，精度和分辨率是两个不同的概念。精度指的是转换后所得结果相对于实际值的准确度，而分辨率指的是能对转换结果发生影响的最小输入量。分辨率很高的A/D转换器可能由于温度漂移、线性不良等原因而并不具有很高的精度。

3）转换时间和转换速度

A/D转换器完成一次转换所需要的时间称为A/D转换时间。转换速度是转换时间的倒数。

2. A/D转换器的选择

1）精度及分辨率的选择

用户提出的数据采集精度是综合精度，可分为传感器精度、信号调节电路精度和A/D转换精度。应将综合精度在各个环节上进行分配，以确定A/D转换器的精度要求，再根据它来确定A/D转换器的位数。

2）转换速度的选择

有的系统有实时性要求。对于快速信号的采集，有时找不到高速的A/D转换芯片，应考虑采用采样/保持电路。

3）转换器输出状态的选择

A/D转换器输出状态的选择包括以下内容：是并行输出还是串行输出，是二进制码输出还是BCD码输出，是用外部时钟、内部时钟还是不用时钟，有无转换结束状态信号，与TTL、CMOS电路的兼容性等。

4）工作温度范围

由于温度会对A/D转换器内部运算放大器和加权电阻网络等产生影响，所以只有在一定的温度范围内才能保证额定精度指标。较好的转换器件的工作温度为－40℃～85℃，较差者为0℃～70℃。

3. A/D转换器的种类

A/D转换器的种类很多，目前最常用的是逐次逼近式A/D转换器和双积分式A/D转换器。

1）逐次逼近式A/D转换器

逐次逼近式A/D转换器是目前种类最多、数量最大、应用最广的A/D转换器件。逐次逼近式A/D转换器中有一个逐次逼近寄存器SAR，数字量是由它产生的。SAR使用对分搜索法产生数字量。以8位数字量为例，SAR首先产生8位数字量的一半，即10000000b，试探模拟量U_i的大小。若$U_o>U_i$，则清除最高位；反之，则保留最高位。在最高位确认后，SAR又以对分搜索法确定次高位，即以7位数字量的一半y1000000b(y由前面的过程已确认)，试探模拟量U_i的大小。依此类推，直到确定了bit0为止，转换结束。

2）双积分式 A/D 转换器

双积分式 A/D 转换器的基本原理是对输入模拟电压和参考电压分别进行两次积分，将输入电压平均值变成与之成正比的时间间隔，然后利用时钟脉冲和计数器测出此时间间隔，进而得到相应的数字量输出。由于该转换电路是对输入电压的平均值进行变换，所以它具有很强的抗工频干扰能力，但双积分式 A/D 转换器转换时间较长，在一些非快速过程的数字测量中得到广泛应用。

4. ADC0809 芯片的工作原理

ADC0809 是采样分辨率为 8 位的、以逐次逼近原理进行模/数转换的器件。其内部有一个 8 通道选择开关，它可以根据地址码锁存译码后的信号，选通 8 路模拟输入信号中的一路进行 A/D 转换。

1）ADC0809 芯片的主要特性

ADC0809 芯片的主要特性如下：

(1) 8 路输入通道，8 位 A/D 转换器，即分辨率为 8 位。

(2) 具有转换启停控制端和转换结束信号输出端。

(3) 输入/输出与 TTL 电平兼容。

(4) 转换时间为 128 μs(与 CLK 引脚外接时钟信号有关)。

(5) 单个＋5 V 电源供电，模拟输入电压范围为 0～＋5 V，不需零点和满刻度校准。

(6) 工作温度范围为－40℃～＋85℃。

(7) 功耗低，约 15 mW。

2）ADC0809 芯片的内部结构

ADC0809 芯片的内部结构如图 7-25 所示，其内部有一个 8 通道多路开关，允许 8 路模拟量分时输入，共用一个 A/D 转换器进行转换。通道地址锁存和译码电路对 ADDA、ADDB、ADDC 三个地址信号进行锁存和译码，其输出用于通道选择。当选中某路并启动 A/D 转换时，该路模拟信号进行 A/D 转换，当 OE 有效时，便可输出转换结果。

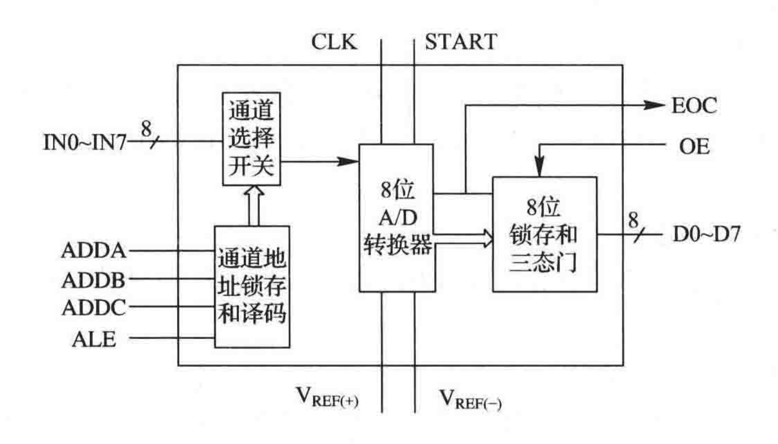


图 7-25　ADC0809 芯片的内部结构

3）ADC0809 芯片的引脚功能

ADC0809 芯片有 28 个引脚，采用双列直插式封装，如图 7-26 所示。

IN0～IN7：8 路模拟量输入端。在多路开关控制下，任一瞬间只能有一路模拟量经相应通道输入到 A/D 转换器中的比较放大器。

D0～D7：8位数字量输出端，可直接接入单片机的数据总线。

ADDA、ADDB、ADDC：3位地址输入线，用于选通8路模拟输入中的某一路。这3位地址线与通道的对应关系见表7-2。

ALE：地址锁存允许信号。该信号的上升沿，可将地址选择信号ADDA、ADDB、ADDC锁入地址寄存器内。

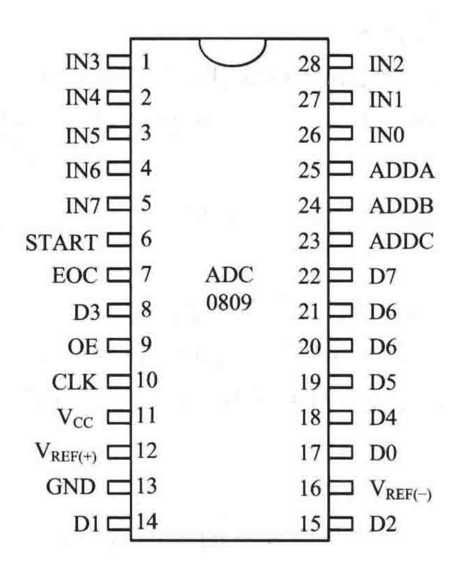


图7-26　ADC0809引脚图

表7-2　通道选择表

ADDC	ADDB	ADDA	选择的通道
0	0	0	IN0
0	0	1	IN1
0	1	0	IN2
0	1	1	IN3
1	0	0	IN4
1	0	1	IN5
1	1	0	IN6
1	1	1	IN7

START：A/D转换启动信号。在START信号的上升沿，将逐次逼近寄存器复位；在START信号的下降沿，开始进行A/D转换；在转换过程中，START保持低电平。

EOC：A/D转换结束信号，输出。EOC=0，表示正在转换(转换期间一直为低电平)；EOC=1，表示A/D转换结束。该信号既可作为查询的状态标志，又可作为中断请求信号使用。

OE：数据输出允许信号，输入，高电平有效，用于控制三态输出锁存器向单片机输出转换得到的数字量。OE=0，输出数据线呈高阻态；OE=1，才能打开三态输出锁存器。

CLK：外部时钟脉冲输入端。ADC0809芯片内部没有时钟电路，故所需时钟信号由外界提供。CLK的频率决定了A/D转换器的转换速度，ADC0809的频率不能高于

640 kHz。

$V_{REF(+)}$ 和 $V_{REF(-)}$：A/D 转换器的参考电压输入线，一般与本机电源和地相连。

V_{CC}：+5 V 电源。

GND：地。

4) ADC0809 芯片的工作过程

ADC0809 芯片的工作过程如下：

(1) 确定 ADDA、ADDB、ADDC 3 位地址，决定选择哪一路模拟信号。

(2) 使 ALE 端接收一正脉冲信号，并使该路模拟信号经选择开关到达比较器的输入端。

(3) 使 START 端接收一正脉冲信号，在 START 的上升沿将逐次逼近寄存器复位，在下降沿启动 A/D 转换。

(4) EOC 输出信号变低，表示转换正在进行。

(5) A/D 转换结束，EOC 变为高电平。此时，数据已保存到 8 位三态输出锁存器中，CPU 可以通过使 OE 信号为高电平，打开 ADC0809 三态输出锁存器，由 ADC0809 输出的数字量传送到 CPU。

5) ADC0809 芯片与单片机的接口

图 7-27 所示为 ADC0809 芯片与单片机的一种接口方法。ADC0809 芯片与单片机接口时，应考虑以下问题：

(1) 8 路模拟通道选择。

一般来说，ADDA、ADDB、ADDC 分别接地址锁存器 74LS373 提供的低 3 位地址，若要选择某个通道，还需输出一个口地址来使 ALE 信号变为高电平，才能将三位地址写入 ADC0809 的通道地址锁存器并译码。图中 ALE 信号是由单片机的 P2.2、P2.1、P2.0 经过 3-8 译码器 74LS138 的输出 $\overline{Y5}$ 与 $\overline{WR}$ 信号"相或"后再经过反相器产生的。因此，ADC0809 的 8 路通道的地址可以确定为 0500H～0507H(P2.2、P2.1、P2.0=101)。

(2) START 信号的产生。

图 7-27 中 ADC0809 芯片的 START 引脚是和 ALE 引脚接在一起的，单片机的 P3.6 ($\overline{WR}$)和 3-8 译码器 74LS138 的输出 $\overline{Y5}$"相或"后再经过反相器接到 ADC0809 芯片的 START、ALE 引脚。因此，只要 $\overline{WR}$ 和 $\overline{Y5}$ 都为 0，反相器的输出信号就会出现高电平。在这个高电平的上升沿，将 ADDA、ADDB、ADDC 地址状态送入地址锁存器中；在下降沿，启动 ADC0809 芯片转换。启动图 7-27 中的 ADC0809 芯片进行转换只需要执行下面几条指令(以通道 0 为例)：

```
MOV     DPTR, #0500H
MOVX    @DPTR, A      ; A 为任意数，MOVX 指令使 WR=0，地址 0500H 使 Y5=0
```

(3) 转换的 CLK 时钟的产生。

ADC0809 的转换频率不能高于 640 kHz。图 7-27 是将单片机的 ALE 引脚经过一个 2 分频电路接在 ADC0809 芯片的 CLK 引脚上的。如果单片机的 f_{osc}=6 MHz，则 ALE 信号的频率为 $2\times f_{osc}/12$，即 1 MHz，再经过 2 分频后，f=500 kHz，满足 ADC0809 芯片的时钟要求。

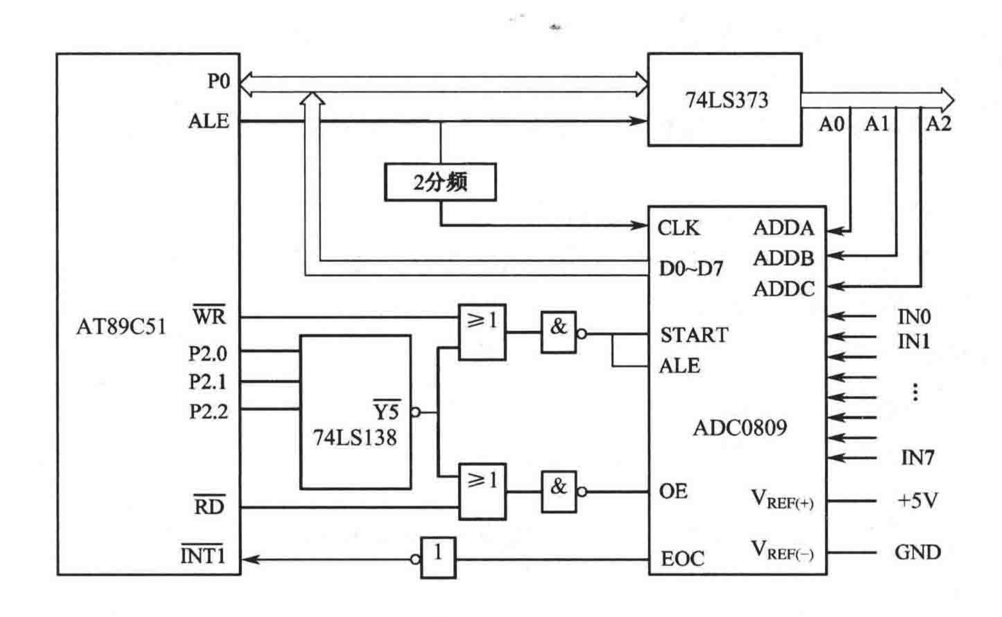


图 7-27　ADC0809 芯片与单片机接口

(4) 如何提供有效的 OE 信号。

图 7-27 是将 P3.7($\overline{RD}$)和 3-8 译码器 74LS138 的输出$\overline{Y5}$"相或"后再经过反相器接到 ADC0809 芯片的 OE 引脚的。因此，只要$\overline{RD}$和$\overline{Y5}$都为 0，OE 引脚就会出现高电平，从而打开三态输出锁存器，A/D 转换的结果也会出现在 P0 口上，输入到单片机中。

(5) CPU 读取 A/D 转换器数据的方式。

① 查询方式。

直接用软件检测 EOC 的状态，EOC=1 时表明 A/D 转换结束，然后进行数据传送。

优点：接口电路设计简单。

缺点：A/D 转换期间独占 CPU，致使 CPU 运行效率降低。

② 定时方式。

ADC0809 芯片的转换时间为 128 μs，我们可设计一个延时 150 μs 的子程序(稍大于 128 μs)，A/D 转换启动后即调用这个子程序，正常情况下延时时间一到，A/D 转换肯定已经完成，接着就可以进行数据传送。

优点：接口电路设计比查询方式的简单，不必读取 EOC 的状态。

缺点：A/D 转换期间独占 CPU，致使 CPU 运行效率降低；另外还必须知道 A/D 转换器的转换时间。

③ 中断方式。

将转换完成的状态信号 EOC 经反相器连接到单片机的外部中断请求引脚。

优点：A/D 转换期间 CPU 可以处理其他的程序，提高了 CPU 的运行效率。

缺点：接口电路复杂。

图 7-27 是将 ADC0809 的 EOC 引脚经过一个反相器接在单片机的$\overline{INT1}$引脚上的，转换结束后，EOC=1，经过反相器后为 0，可以向单片机发出中断请求，也可以作为查询转换结束的标志。

5. ADC0809 芯片的应用举例

【例 7-9】 在图 7-27 的基础上，设计一个 8 路模拟量输入的巡回检测系统。编写程

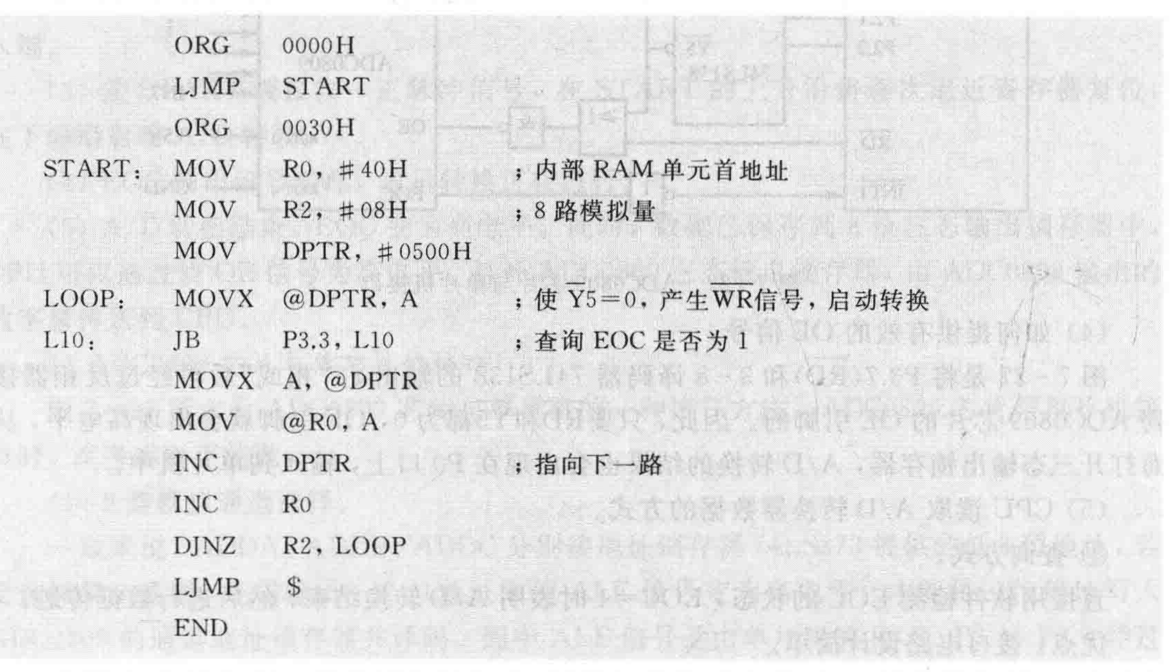
序，分别采用查询方式和中断方式，将采样转换后的数据依次存放在内部 RAM 的 40H～47H 单元中。

分析 1(查询方式)：通过判断 P3.3 的电平变化确定 A/D 转换是否完成，完成后读取数据。

IN0～IN7 的地址是 0500H～0507H。

A/D 转换启动方法是向对应的地址写入一个任意数，这时 $\overline{WR}$为低电平，$\overline{Y5}$为低电平，ADC0809 的 START 输入高电平，A/D 转换开始。

参考程序(查询方式)：



```
        ORG     0000H
        LJMP    START
        ORG     0030H
START:  MOV     R0, #40H        ；内部 RAM 单元首地址
        MOV     R2, #08H        ；8 路模拟量
        MOV     DPTR, #0500H
LOOP:   MOVX    @DPTR, A        ；使 Y5=0，产生WR信号，启动转换
L10:    JB      P3.3, L10       ；查询 EOC 是否为 1
        MOVX    A, @DPTR
        MOV     @R0, A
        INC     DPTR            ；指向下一路
        INC     R0
        DJNZ    R2, LOOP
        LJMP    $
        END
```

分析 2(中断方式)：其他同查询方式，只是转换完成信号 EOC 触发外中断 1，读取数据等操作在中断服务程序中进行。

参考程序(中断方式)：

```
        ORG     0000H
        LJMP    START
        ORG     0013H
        LJMP    INT01
        ORG     0030H
START:  MOV     R0, #40H
        MOV     R2, #08H
        SETB    IT1
        SETB    EX1
        SETB    EA              ；开中断
        MOV     A, #00H
        MOV     DPTR, #0500H
        MOVX    @DPTR, A
HERE:   LJMP    HERE            ；等待中断到来
```

```
INT01:    MOVX   A, @DPTR      ; A/D 转换结束
          MOV    @R0, A
          DJNZ   R2, LOOP
          CLR    EA            ; 转换结束关中断
          LJMP   EXIT
LOOP:     INC    DPTR
          INC    R0
          MOVX   @DPTR, A      ; 再次启动转换
EXIT:     RETI
          END
```

7.5.2 D/A 转换器

1. D/A 转换器概念

D/A 转换器(DAC)是一种将数字信号转换成模拟信号的器件，为计算机系统的数字信号和模拟环境的连续信号之间提供了一种接口。D/A 转换器的输出是由数字输入和参考源 V_{REF} 组合进行控制的。大多数常用的 D/A 转换器的数字输入是二进制码或 BCD 码，输出可以是电流或电压，但多数是电流。因而，在多数电路中，D/A 转换器的输出需要由运算放大器组成的电流/电压转换器将电流输出转换成电压输出。目前 DAC 除了使用并行方式与单片机连接外，还可使用 I^2C 总线和 SPI 串行方式与单片机连接。

1) D/A 转换原理

数字量是用代码按数位组合起来表示的，对于有权码，每位代码都有一定的权值。为了将数字量转换成模拟量，必须将每 1 位的代码按其权值的大小转换成相应的模拟量，然后将这些模拟量相加，即可得到与数字量成正比的总模拟量，从而实现数字—模拟转换。这就是构成 D/A 转换器的基本思路。

在单片集成 D/A 转换器中，使用最多的是倒 T 形电阻网络 D/A 转换器。4 位(D_3 D_2 D_1 D_0)倒 T 形电阻网络 D/A 转换器原理图如图 7-28 所示。

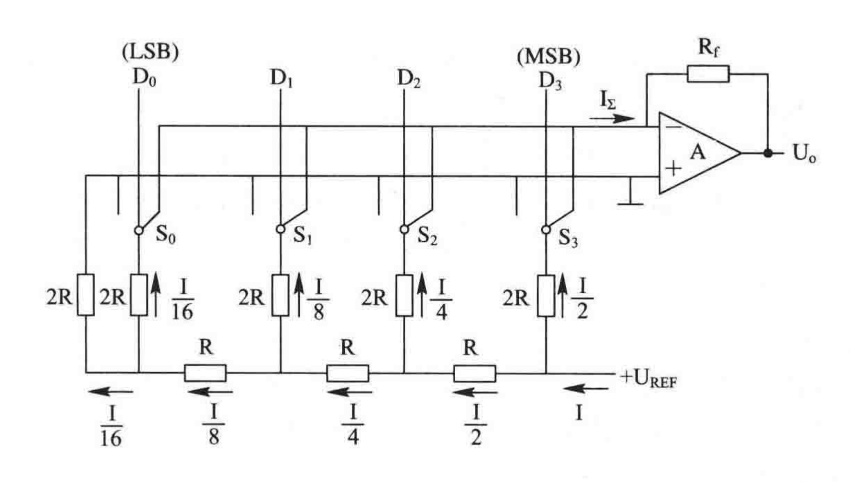


图 7-28 4 位倒 T 形电阻网络 D/A 转换器原理图

S_0～S_3 为模拟开关，R-2R 电阻解码网络呈倒 T 形，运算放大器 A 构成求和电路。S_i

由输入数码 D_i 控制。当 $D_i=1$ 时，S_i 接运放反相输入端（“虚地”），I_i 流入求和电路；当 $D_i=0$ 时，S_i 将电阻 2R 接地。

无论模拟开关 S_i 处于何种位置，与 S_i 相连的 2R 电阻均等效接“地”（地或虚地），这样流经 2R 电阻的电流与开关位置无关，为确定值。

分析 R－2R 电阻解码网络不难发现，从每个接点向左看的二端网络等效电阻均为 R，流入每个 2R 电阻的电流从高位到低位按 2 的整倍数递减。设由基准电压源提供的总电流为 $I(I=U_{REF}/R)$，则流过各开关支路（从右到左）的电流分别为 I/2、I/4、I/8 和 I/16。

流过电阻 R_f 的电流为

$$I_{\sum}=\frac{U_{REF}}{R}\cdot\left(\frac{D_0}{2^4}+\frac{D_1}{2^3}+\frac{D_2}{2^2}+\frac{D_3}{2^1}\right)=\frac{U_{REF}}{16R}\cdot\sum_{i=0}^{3}2^i\cdot D_i \tag{7-1}$$

又 $U_o=-R_f\cdot I_{\sum}$，故输出电压为

$$U_o=-R_f\cdot\frac{U_{REF}}{16R}\cdot\sum_{i=0}^{3}2^i\cdot D_i \tag{7-2}$$

将输入数字量扩展到 n 位，可得 n 位数字量倒 T 形电阻网络 D/A 转换器输出模拟量与输入数字量之间的一般关系式：

$$U_o=-\frac{R_f}{R}\cdot\frac{U_{REF}}{2^n}\cdot\sum_{i=0}^{n-1}2^i\cdot D_i \tag{7-3}$$

2）D/A 转换芯片的主要性能指标

（1）分辨率。

D/A 转换器的分辨率是指当输入数字是发生单位数码（1LSB）变化时，所对应输出模拟量（电压或电流）的变化量。对于线性 D/A 转换器来说，分辨率＝模拟输出的满量程值/2^n（其中 n 为数字量位数）。在实际使用中，常采用输入数字量的位数或最大输入码的个数来表示分辨率。例如，8 位 D/A 转换器，其分辨率为 8 位。显然，位数越多，分辨率就越高。

（2）线性度。

D/A 转换器的线性度用非线性误差的大小来表示。非线性误差是指理想的输入/输出特性的偏差与满刻度输出之比的百分数。

（3）精度。

D/A 转换器的精度是实际输出电压与理论输出电压相差程度的一个量度，与 D/A 转换芯片的结构和接口配置的电路有关。一般来说，当不考虑其他 D/A 转换误差时，D/A 转换的精度即为分辨率的大小。故要获得高精度的 D/A 转换结果，首先要保证选择有足够分辨率的 D/A 转换器。但是 D/A 转换精度还与外电路的配置有关，当外电路的器件或电源误差较大时，会造成较大的 D/A 转换误差。当这些误差超过一定程度时，会使增加 D/A 转换位数失去意义。

（4）转换时间。

D/A 转换器的转换时间又称建立时间，一般是指在输入的数字量发生变化后，输出的模拟量稳定到相应数值范围内（稳定值±1/2 LSB）所经历的时间。

（5）标称满量程和实际满量程。

标称满量程（NFS）是指对应于数字量标称值 2^n 的模拟输出量。但实际数字量最大为 2^n-1，比标称值小 1 个 LSB，因此实际满量程（AFS）比标称满量程（NFS）小 1 个 LSB 增

量，即 AFS＝NFS－1LSB 增量＝$(2^n-1)/2^n \times NFS$。

2. DAC0832 芯片的工作原理及应用

1）DAC0832 的主要特性

DAC0832 是采用 CMOS/Si-Cr 工艺制成的双列直插式单片 8 位 D/A 转换器。它可直接与 MCS－51 单片机相连，以电流形式输出；当转换为电压输出时，可外接运算放大器。其主要应用特性如下：

（1）分辨率为 8 位，建立时间为 1 μs，功耗为 20 mW。

（2）DAC0832 芯片是与微处理器兼容的 D/A 转换器，逻辑电平输入与 TTL 兼容，芯片的许多控制引脚可以和微处理器的控制线直接相连，接受微处理器控制。

（3）数字输入端具有双重锁存控制功能，可以双缓冲、单缓冲或直通数字输入，实现多通道 D/A 的同步转换输出。

（4）芯片内部没有参考电压，必须外接参考电压电路。

（5）该芯片为电流输出型 D/A 转换器，要获得模拟电压输出，需外加转换电路。

2）DAC0832 的内部结构及外部引脚

DAC0832 由输入（数据）寄存器、DAC 寄存器和 D/A 转换器三大部分组成，如图 7－29所示。

DAC0832 内部采用 R－2R 梯形电阻网络。输入寄存器和 DAC 寄存器用以实现两次缓冲，故在输出的同时，还可同时存一个数字，从而提高了转换速度。当多芯片同时工作时，可用同步信号实现各模拟量同时输出。

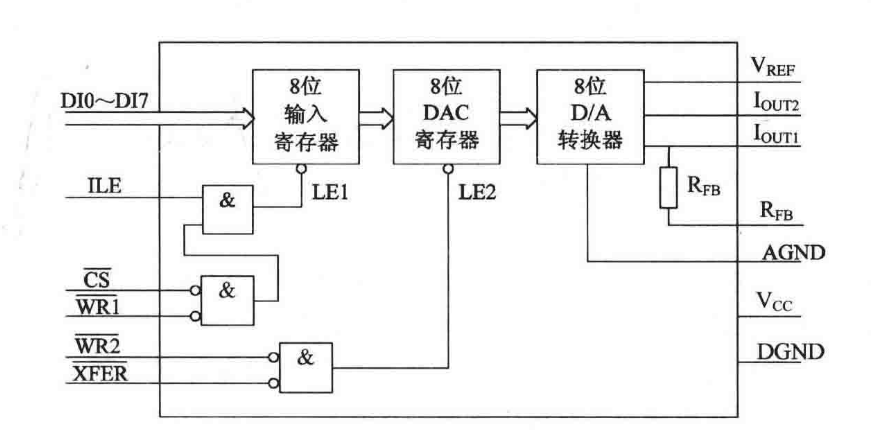


图 7－29　DAC0832 的内部结构

$\overline{CS}$：片选信号，低电平有效。与 ILE 相配合，可对写信号$\overline{WR1}$是否有效起到控制作用。

ILE：允许输入锁存信号，高电平有效。输入寄存器的锁存信号 LE1 由 ILE、$\overline{CS}$、$\overline{WR1}$的逻辑组合产生。当输入寄存器的锁存信号为高电平时，输入寄存器与数据线上的状态一致，输入寄存器的锁存信号的负跳变将输入在数据线上的信息存入输入锁存器。

$\overline{WR1}$：写信号 1，低电平有效。当$\overline{WR1}$、$\overline{CS}$、ILE 均有效时，可将数据写入 8 位输入寄存器。

$\overline{WR2}$：写信号 2，低电平有效。当$\overline{WR2}$有效时，在$\overline{XFER}$传送控制信号作用下，可将锁存在输入寄存器的 8 位数据送到 DAC 寄存器。

$\overline{XFER}$：数据传送信号，低电平有效。当$\overline{WR2}$、$\overline{XFER}$均有效时，在 DAC 寄存器的锁存信号产生正脉冲，当 DAC 寄存器的锁存信号为高电平时，DAC 寄存器的输出和输入寄存器的状态一致，DAC 寄存器的锁存信号负跳变，输入寄存器的内容存入 DAC 寄存器。

V_{REF}：基准电源输入端。它与 DAC 内的 R－2R 梯形网络相接，V_{REF}可在±10 V 范围内调节。

DI0～DI7：8 位数字量输入端。DI7 为最高位，DI0 为最低位。

I_{OUT1}：DAC 的电流输出 1。当 DAC 寄存器各位为 1 时，输出电流为最大；当 DAC 寄存器各位为 0 时，输出电流为 0。

I_{OUT2}：DAC 的电流输出 2。I_{OUT1} + I_{OUT2}恒为一常数。

R_{FB}：反馈电阻。在 DAC0832 芯片内有一个反馈电阻，所以，R_{FB}端可以直接接到外部运算放大器的输出端，相当于将反馈电阻接在运算放大器的输入端和输出端之间。

V_{CC}：电源输入线。

DGND：数字地。

AGND：模拟信号地。

3）DAC0832 的工作方式

用 DAC0832 芯片实现 D/A 转换有 3 种方式，即直通方式、单缓冲方式和双缓冲方式。

（1）直通方式。

直通方式就是使 DAC0832 的两个寄存器均处于直通状态，因此要将$\overline{CS}$、$\overline{WR1}$、$\overline{WR2}$和$\overline{XFER}$端都接数字地，ILE 接高电平，数据直接送入 D/A 转换电路进行 D/A 转换。这种方式可用于一些不采用微机的控制系统中。

（2）单缓冲方式。

单缓冲方式就是使 DAC0832 芯片的两个寄存器之一处于直通状态，另一个处于寄存器锁存状态。这时只需执行一次写操作，打开锁存的寄存器，即可使数字量通过输入寄存器和 DAC 寄存器完成 D/A 转换。

第一种方法是使输入寄存器工作在锁存状态，而 DAC 寄存器工作在直通状态，如图 7－30(a)所示。具体地说，就是使$\overline{WR2}$和$\overline{XFER}$都为低电平，DAC 寄存器的锁存选通端得不到有效电平而直通；此外，使输入寄存器的控制信号 ILE 处于高电平、$\overline{CS}$处于低电平，这样，当$\overline{WR1}$端来一个负脉冲时，就可以完成一次转换。

第二种方法是使输入寄存器工作在直通状态，而 DAC 寄存器工作在锁存状态，如图 7－30(b)所示。就是使$\overline{WR1}$和$\overline{CS}$为低电平，ILE 为高电平，这样，输入寄存器的锁存选通信号处于无效状态而直通；当$\overline{WR2}$和$\overline{XFER}$端输入 1 个负脉冲时，DAC 寄存器工作在锁存状态，提供锁存数据进行转换。

【例 7－10】 如图 7－30(a)所示，用 DAC0832 输出 0～＋5 V 三角波，电路为单缓冲方式。设 U_{REF}＝－5 V，DAC0832 地址为 7FFFH。

分析：利用“INC　A”，A 中的值可以加到 FFH，直到 00H，这时，通过“JNZ　UP”结束三角波上升段；这时 A 中的值是 00H，下降段程序第 1 条语句是“DEC　A”，执行后 A 中的值为 FFH，以后再顺序减 1，直到 A 中的值为 00H，通过“JNZ　DOWN”结束三角波下降段。

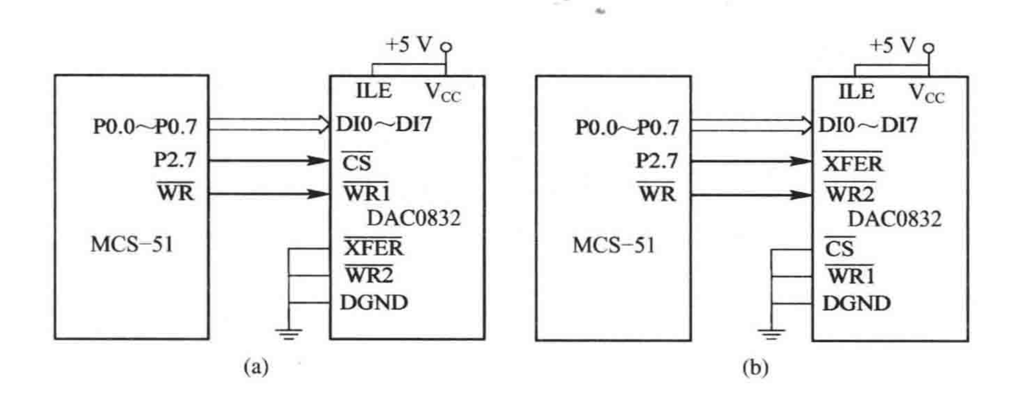
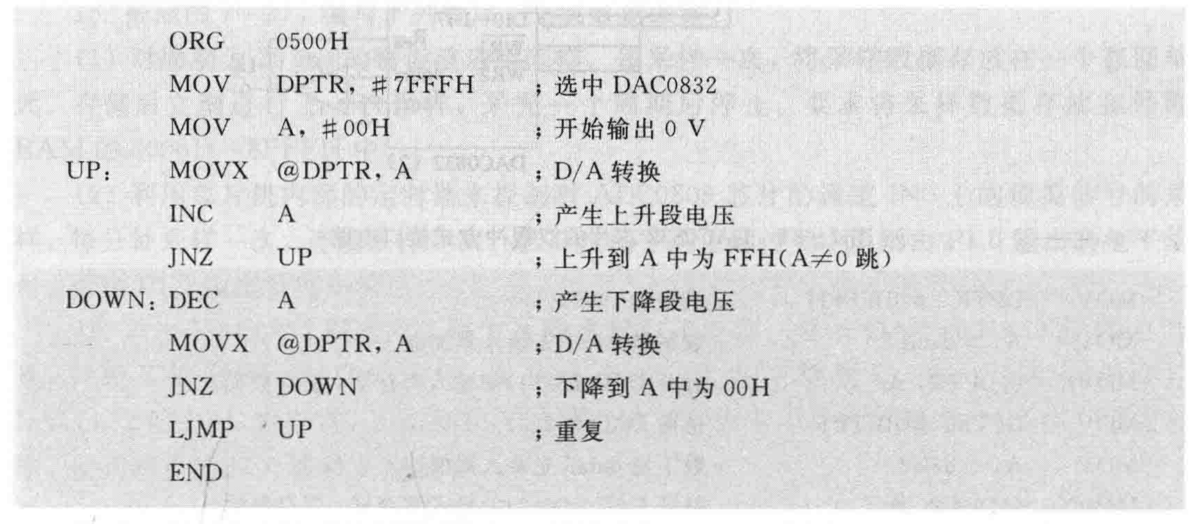


图 7－30　DAC0832 芯片的单缓冲方式接口电路

参考程序：



```
        ORG     0500H
        MOV     DPTR，#7FFFH      ；选中 DAC0832
        MOV     A，#00H           ；开始输出 0 V
UP：    MOVX    @DPTR，A          ；D/A 转换
        INC     A                 ；产生上升段电压
        JNZ     UP                ；上升到 A 中为 FFH(A≠0 跳)
DOWN：  DEC     A                 ；产生下降段电压
        MOVX    @DPTR，A          ；D/A 转换
        JNZ     DOWN              ；下降到 A 中为 00H
        LJMP    UP                ；重复
        END
```

注意：若想改变波形的周期(频率)，只需在“JNZ　UP”和“JNZ　DOWN”前插入延时程序即可。

(3) 双缓冲方式。

对于多路 D/A 转换接口，要求同步进行 D/A 转换输出时，必须采用双缓冲方式接法。采用这种接法时，数字量的输入锁存和 D/A 转换输出是分以下两步完成的：

① CPU 分时向各路 D/A 转换器输入要转换的数字量并锁存在各自的输入寄存器中。

② CPU 对所有的 D/A 转换器发出控制信号，使各路输入寄存器中的数据进入 DAC 寄存器，实现同步转换输出。

因此，双缓冲方式特别适用于要求同时输出多个模拟量的场合。此时需要采用多片 D/A 转换器芯片，每片控制 1 个模拟量的输出。

图 7－31 所示是 DAC0832 的双缓冲方式接口电路示意图。根据图中的线路连接，得知 DAC0832(1)的输入寄存器口地址为 DFFFH，DAC0832(2)的输入寄存器口地址为 BFFFH，两个 DAC0832 的 DAC 寄存器口地址同为 7FFFH。下面几条指令可完成两种数字量到模拟量的同步转换。

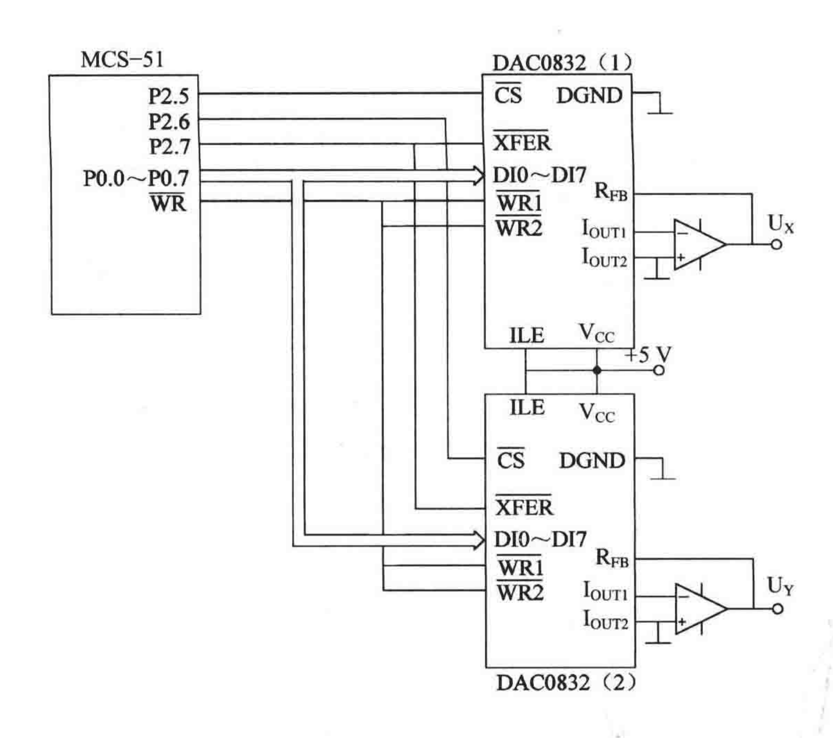


图 7-31 DAC0832 芯片的双缓冲方式接口电路

```
MOV    DPTR, #0DFFFH      ；指向 DA0832(1)
MOV    A, #data1          ；数字量 data1 先装入累加器
MOVX   @DPTR, A           ；打开 DAC0832(1)的输入寄存器，锁存数据
MOV    DPTR, #0BFFFH      ；指向 DAC0832(2)
MOV    A, #data2          ；数字量 data2 先装入累加器
MOVX   @DPTR, A           ；打开 DAC0832(2)的输入寄存器，锁存数据
MOV    DPTR, #7FFFH       ；同时指向两个 DAC0832 的 DAC 寄存器
MOVX   @DPTR, A           ；在WR有效时，完成一次 D/A 输入并转换
```

注意：执行最后“MOVX　@DPTR，A”这条指令时，实际上与 A 中数据是多少没有关系，仅利用执行 MOVX 指令时出现的写信号 $\overline{WR}$ 打开 DAC 寄存器。

思考与练习

1. 在 MCS-51 单片机扩展系统中，外部程序存储器和外部数据存储器共同使用了 16 位地址线和 8 位数据线，为什么这两个存储空间不会发生冲突？

2. 键盘、开关的抖动产生原因是什么？有什么办法消除？

3. 独立式按键和矩阵式按键分别有什么特点？适用于什么场合？

4. 共阴极 LED 数码管和共阳极 LED 数码管在应用中有何区别？选择的原则是什么？

5. LED 数码管的静态显示和动态显示有什么不同？优缺点是什么？实际设计时应如何选择？

6. 在动态扫描显示电路中，为什么可以不接限流电阻？

7. 设计 ADC0809 与单片机的接口时，要用到哪些控制信号？它们的作用是什么？

8. 画出 ADC0809 采用查询方式与 8031 的接口电路，并编制程序。

9. 设计 DAC0832 与单片机的接口时，要用到哪些控制信号？它们的作用是什么？

10. 使用 DAC0832 时的单缓冲方式和双缓冲方式是如何工作的？它们各占用外部 RAM 的几个单元？在软件编程上有什么区别？

11. 试将 8031 单片机外接一个 2716 EPROM 和一片 6116 RAM 组成一个应用系统，画出硬件连接图，并指出扩展存储器的地址范围。

12. 有 8 片 8 KB 的 RAM 芯片，用 74LS138 进行地址译码，实现 MCS-51 的数据存储器的扩展，要求画出连接图，并说明各芯片的地址范围。

13. 设计一个 3×3 的矩阵键盘并叙述其工作原理。

14. 编写图 7-18 所示的动态扫描显示电路的显示程序，使电路中的 LED 轮流显示"123456"和"PLEASE"，每隔一段时间切换一次。

15. 按照图 7-27，编写下列程序，完成数据采集：

(1) 对周期为 25 ms 的锯齿波进行采样。每采样一次，将采样数据存放在一个存储单元。存储后立刻进行下一次采样，采完一个周期后停止。要求将采样数据存放在外部 RAM 的 8000H～8FFFH 中。

(2) 利用单片机内部的定时器来控制对 ADC0809 芯片的通道 IN0 上的模拟信号的采样，每分钟采样一次，连续采样 5 次。若 5 次的平均值超过 80H，则由 P1.0 输出高电平信号，否则 P1.0 输出低电平信号。

16. 画出 DAC0832 双缓冲工作方式的典型应用电路。要求 DAC 0832(1)由 P2.0 片选，代表 X 轴信号，DAC 0832(2)由 P2.1 片选，代表 Y 轴信号，P2.2 同时控制两片 DAC0832 的 DAC 寄存器，X 信号和 Y 信号已存在片内 RAM 的 40H 和 41H 中。编写程序，使其同步输出 X 轴和 Y 轴信号。

17. 用 DAC0832 芯片，设计有 3 路模拟量同时输出的 MCS-51 系统，画出硬件结构框图，编写数/模转换程序。

第8章　单片机串行扩展技术

本章主要介绍 I^2C 总线和 SPI 总线的原理、时序和扩展方法，并详细讲解单片机系统中最典型的串行总线器件的应用。本章所述的基本原理和时序控制可以推广到其他类似总线标准的器件。

8.1　串行总线概述

近年来，由于集成电路芯片技术的进步，单片机应用系统越来越多地采用串行总线进行扩展。

与并行总线相比，采用串行总线进行扩展时，简化了系统的连线，缩小了电路板的面积，节省了系统的资源，具有扩展性好、成本低廉、可靠性高、硬件易于模块化等优点。因此，采用串行总线扩展方法是当前单片机应用系统设计的流行趋势。

目前单片机应用系统常用的串行扩展总线有单总线、I^2C 总线(二线总线)和 SPI 总线(三线总线)。

8.2　I^2C 总线接口及其扩展

I^2C(Inter Integrated Circuit)总线是 Philips(飞利浦)公司推出的芯片间串行数据传输总线，后来发展成为嵌入式系统设备间通信的国际标准，用于连接微控制器及其外围设备，是微电子通信控制领域广泛采用的一种总线标准。它是同步通信的一种特殊形式，具有接口线少，控制方式简单，器件封装形式小，通信速率较高等优点。

8.2.1　I^2C 总线的基础知识

1. I^2C 总线的特点

I^2C 总线的特点如下：

(1) 采用二线制。采用二线制连接，可以减少器件的引脚，简化器件间连接电路的设计，有效减小电路板的体积，提高系统的可靠性和灵活性。

(2) 传输速率高。标准模式的传输速率为 100 kb/s，快速模式为 400 kb/s，高速模式为 3.4 Mb/s。

(3) 支持多主和主/从两种工作方式。多主方式时，要求各主单片机配备 I^2C 总线标准接口；而基本型 AT89C51 或 80C51 单片机没有 I^2C 总线标准接口，只能工作于主/从方式(扩展外围器件)。本节仅介绍主/从方式，并将单片机称为主机，扩展的器件称为从机。

2. I^2C 总线的架构

I^2C 总线只有两根连线。一根是数据线 SDA，另一根是时钟线 SCL。所有连接到 I^2C

总线上的器件的数据线都接到 SDA 上，各器件的时钟线均接到 SCL 上。I^2C 总线的基本架构如图 8－1 所示。

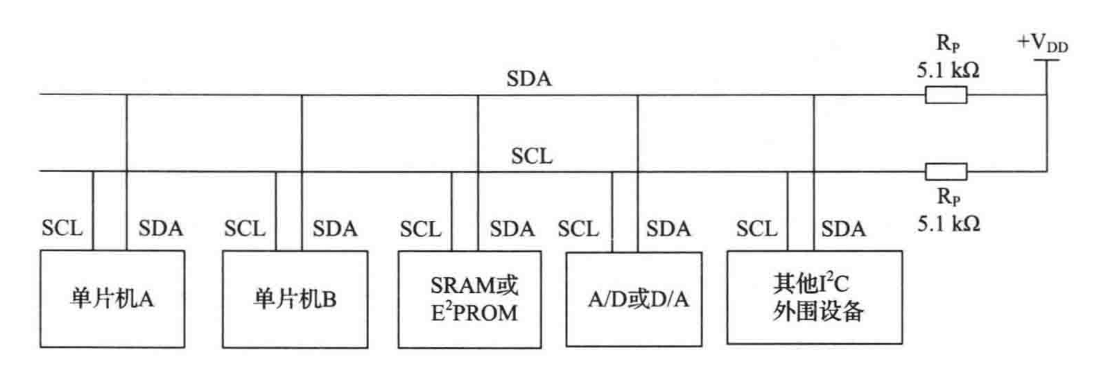


图 8－1 I^2C 总线的基本架构

3. I^2C 总线的常用器件

I^2C 总线广泛用于各种新型芯片中，如 I/O 电路、存储器、A/D 转换器、D/A 转换器、温度传感器及微控制器等。许多器件生产厂商都采用了 I^2C 总线设计产品，如 Atmel 公司的 E^2PROM 器件 AT24C04/08/16/64/128/256/512；MAXIMUM 公司的 A/D 转换器件 MAX1036～MAX1039 等。

8.2.2 I^2C 总线的数据传输时序

在 I^2C 总线上，每一位数据位的传输都与时钟脉冲相对应。逻辑 0 和逻辑 1 的信号电平取决于相应的电源电压。使用不同的半导体制造工艺，如 CMOS、NMOS 等类型的电路都可以接入总线。对于数据传输，I^2C 总线协议规定了如下信号及时序。

1. 起始和停止信号

起始和停止信号如图 8－2 所示。

SCL 为高电平期间，SDA 由高电平向低电平的变化表示起始信号。

SCL 为高电平期间，SDA 由低电平向高电平的变化表示停止信号。

总线空闲时，SCL 和 SDA 两条线都是高电平。SDA 的起始信号和停止信号由主机发出。在起始信号后，总线处于被占用的状态；在停止信号后，总线处于空闲状态。

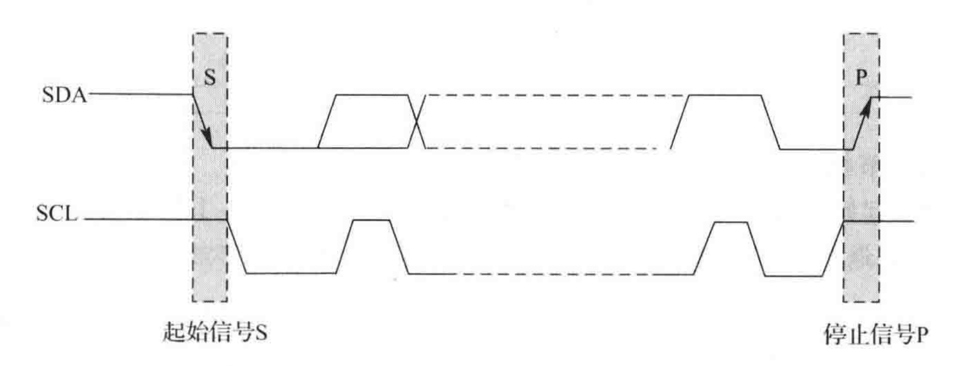


图 8－2 起始和停止信号

2. 字节传输时序

传输的每个字节必须是 8 位长度。先传最高位(MSB)，每个被传输字节后面都要跟随

应答位(即一帧共有 9 位)，如图 8－3 所示。

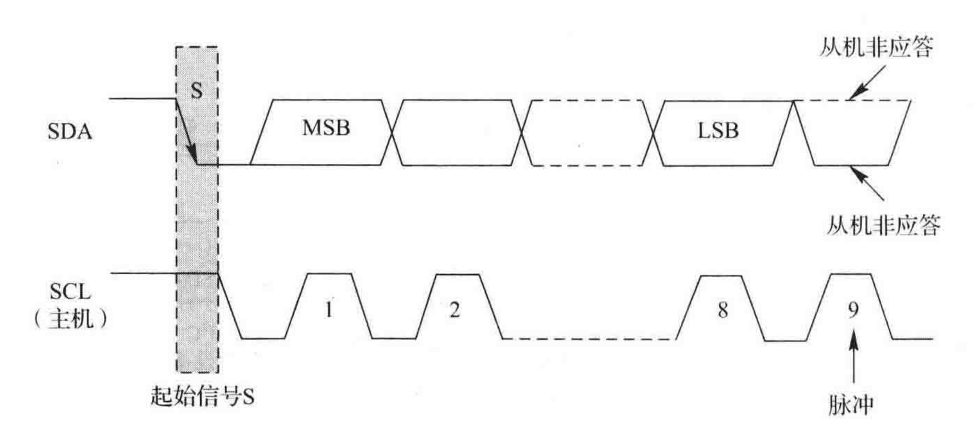


图 8－3　字节传输时序

从机接收数据时，在第 9 个时钟脉冲要发出应答脉冲，但在数据传输一段时间后无法接收更多的数据时，从机可以采用"非应答"通知主机，主机在第 9 个时钟脉冲检测到 SDA 线无有效应答负脉冲(即非应答)时会发出停止信号，以结束数据传输。

与主机发送数据相似，主机在接收数据时，它收到最后一个数据字节后，必须向从机发出一个结束传输的"非应答"信号。然后从机释放 SDA 线，以允许主机产生停止信号。

3. 数据传输时序

对于数据(多字节)传输，I^2C 总线协议规定：SCL 由主机控制，从机在自己忙时拉低 SCL 线以表示自己处于"忙状态"。字节数据由主机发出，响应位由从机发出。SCL 高电平期间，SDA 线数据要稳定；SCL 低电平期间，SDA 线数据允许更新。数据传输时序如图 8－4所示。

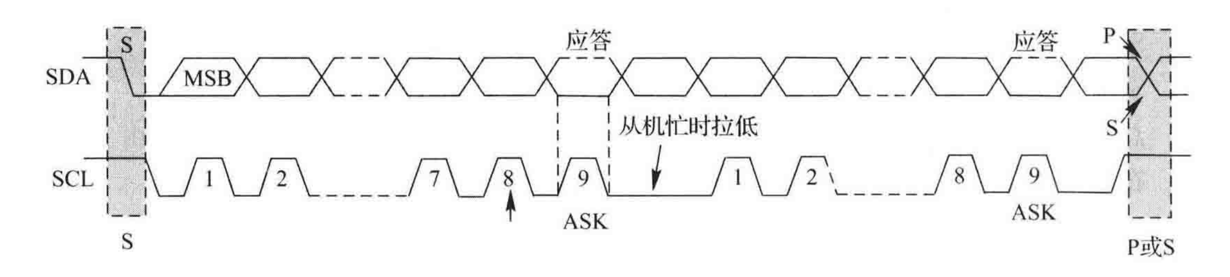


图 8－4　数据传输时序

4. 寻址字节格式

主机发出起始信号后要先传送 1 个寻址字节：7 位从机地址，1 位传输方向控制位(用"0"表示主机发送数据，"1"表示主机接收数据)，格式如下：

位	D7	D6	D5	D4	D3	D2	D1	D0
	DA3	DA2	DA1	DA0	A2	A1	A0	$R/\overline{W}$

D7～D1 组成从机的地址。D0 是数据传送方向位，$R/\overline{W}$ 确定从机下一字节数据是读

出还是写入(0 为写入，1 为读出)。主机发送地址时，总线上的每个从机都将这 7 位地址码与自己的地址进行比较。如果相同，则认为自己正被主机寻址。其中：D7～D4 为器件固有地址编码，固定为 1010；D3～D1 为器件引脚编码 A2、A1、A0，正好与芯片的 3、2、1 引脚对应，为当前电路中的地址选择线，三根线可选择 8 个芯片同时连接在电路中，当要与哪个芯片通信时传送相应的地址即可与该芯片建立连接。

I^2C 器件地址由固定部分和可编程部分组成。

8.2.3　I^2C 总线的时序模拟

对于没有配置 I^2C 总线接口的单片机(如 80C51、AT89C51 等)，可以利用通用并行 I/O口线模拟 I^2C 总线接口的时序。

1. 典型信号的时序

I^2C 总线的数据传输有严格的时序要求。I^2C 总线的起始信号、停止信号、发送应答“0”及发送非应答“1”的时序如图 8-5 所示。

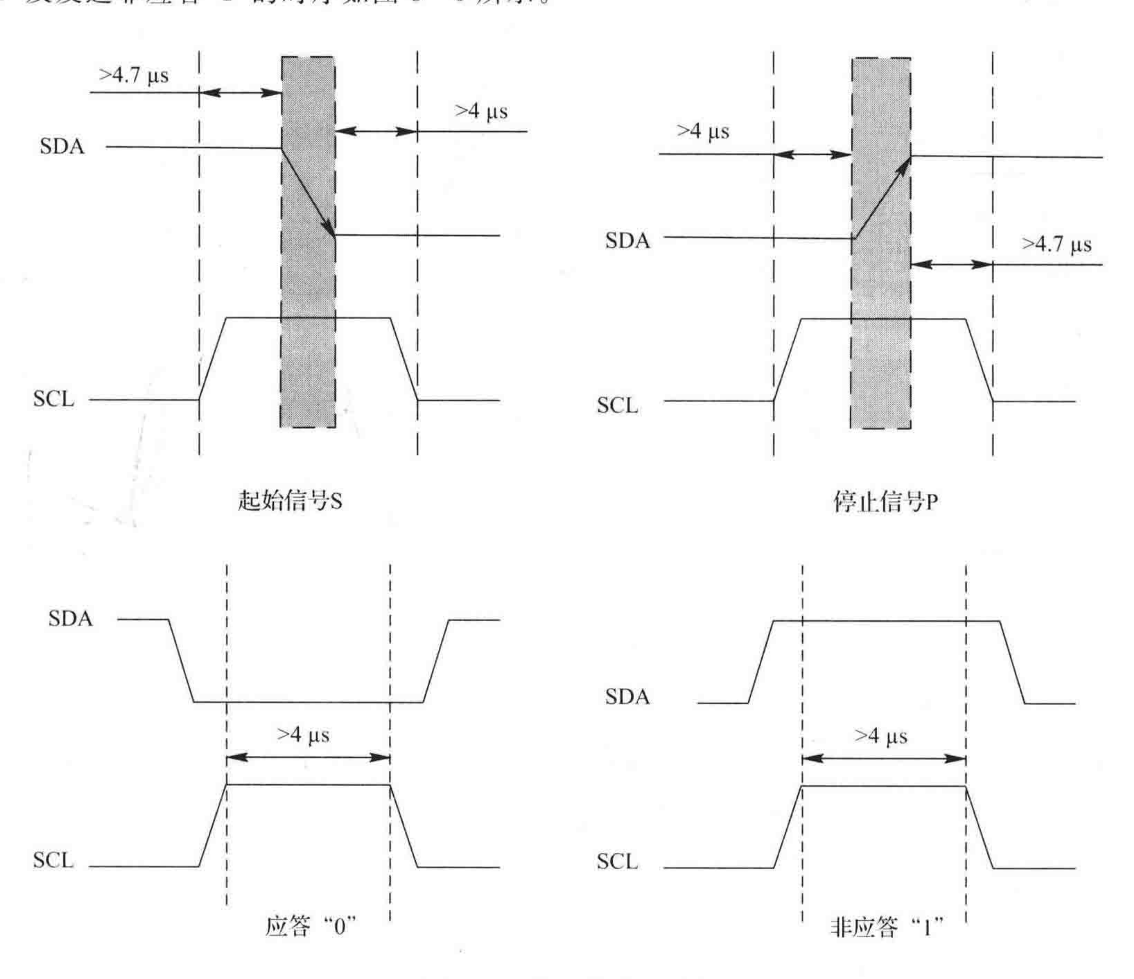


图 8-5　典型信号的时序

2. 典型信号模拟子程序

设主机采用 AT89C51 单片机，下面给出几个典型信号的模拟子程序。

【例 8-1】　AT89C51 的 P2.0 模拟数据线 SDA，P2.1 模拟时钟线 SCL，晶振频率为 12 MHz，编写 I^2C 总线的起始信号、应答“0”时序的程序段。

参考程序：

起始伪指令：

```
SDA    EQU    P2.0     ；用 SDA 代表 P2.0，数据线
SCL    EQU    P2.1     ；用 SCL 代表 P2.1，时钟线
```

延时子程序：

```
DELAY：NOP           ；延时子程序，延时 6 μs 左右，按要求大于 4.7 μs
       NOP
       NOP
       NOP
       NOP
       RET
```

起始信号：

```
SETB    SDA     ；SDA 置 1
SETB    SCL     ；SCL 置 1
LCALL   DELAY
CLR     SDA     ；SDA 清 0
LCALL   DELAY
CLR     SCL     ；SCL 清 0
```

应答“0”：

```
CLR     SDA     ；SDA 清 0
SETB    SCL     ；SCL 置 1
LCALL   DELAY
CLR     SCL     ；SCL 清 0
SETB    SDA     ；SDA 置 1
```

8.2.4 串行程序存储器 AT24C04

串行 E^2PROM 的优点是体积小，功耗低，占用 I/O 口线少，性能价格比高。Atmel 公司的 E^2PROM 是一个系列，即 AT24CXX 系列存储器器件。典型产品是 AT24C04，内含 512 B，擦写次数大于 100 万次，写入周期不大于 10 ms。

1. AT24C04 的引脚定义

AT24C04 引脚图如图 8－6 所示。

A0～A2：地址线。

SDA：数据输入/输出线。

SCL：串行时钟线。

WP：写保护控制端，接地时允许写入。

AT24C04 与单片机的连接如图 8－7 所示。

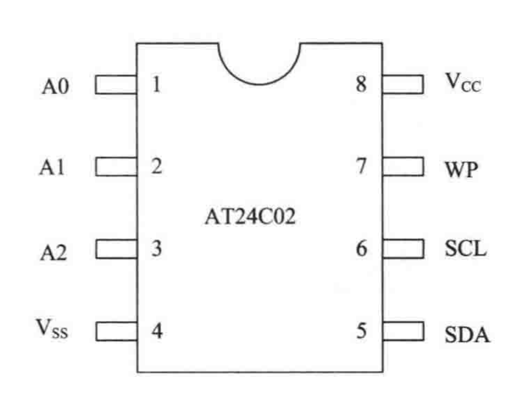


图 8－6　AT24C04 引脚图

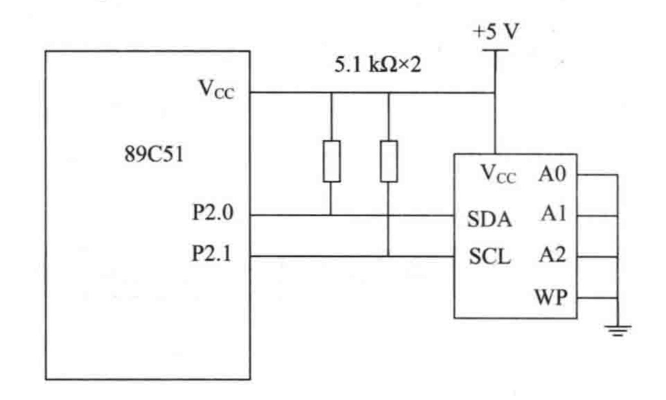


图 8－7　AT24C04 与单片机的连接

2. AT24CXX 系列存储器器件的地址

I^2C 器件地址由固定部分和可编程部分组成，即 AT24CXX 系列存储器器件的地址如表 8－1 所示。以 AT24C04 为例，器件地址的固定部分为 1010，引脚 A2 和 A1 的组合可以选择 4 个同样的器件。片内 512 个字节单元的访问，由第 1 字节(器件寻址字节)的 P0 位及下一字节(8 位的片内储存地址选择字节)共同寻址。

表 8－1　AT24CXX 系列存储器器件的地址

<table>
<tr><th rowspan="2">器件型号</th><th rowspan="2">字节容量</th><th colspan="8">器件寻址字节</th><th rowspan="2">内部地址字节数</th><th rowspan="2">页面写字节数</th><th rowspan="2">最多可接器件数</th></tr>
<tr><th colspan="4">固定标示</th><th colspan="3">片选</th><th>R/$\overline{W}$</th></tr>
<tr><td>AT24C01A</td><td>128 B</td><td rowspan="10">1</td><td rowspan="10">0</td><td rowspan="10">1</td><td rowspan="10">0</td><td>A2</td><td>A1</td><td>A0</td><td>1/0</td><td rowspan="5">1</td><td>8</td><td>8</td></tr>
<tr><td>AT24C02</td><td>256 B</td><td>A2</td><td>A1</td><td>A0</td><td>1/0</td><td>8</td><td>8</td></tr>
<tr><td>AT24C04</td><td>512 B</td><td>A2</td><td>A1</td><td>P0</td><td>1/0</td><td>16</td><td>4</td></tr>
<tr><td>AT24C08A</td><td>1 KB</td><td>A2</td><td>A1</td><td>P0</td><td>1/0</td><td>16</td><td>2</td></tr>
<tr><td>AT24C16A</td><td>2 KB</td><td>A2</td><td>A1</td><td>P0</td><td>1/0</td><td>16</td><td>1</td></tr>
<tr><td>AT24C32A</td><td>4 KB</td><td>A2</td><td>A1</td><td>A0</td><td>1/0</td><td rowspan="5">2</td><td>32</td><td>8</td></tr>
<tr><td>AT24C64A</td><td>8 KB</td><td>A2</td><td>A1</td><td>A0</td><td>1/0</td><td>32</td><td>8</td></tr>
<tr><td>AT24C128B</td><td>16 KB</td><td>A2</td><td>A1</td><td>A0</td><td>1/0</td><td>64</td><td>8</td></tr>
<tr><td>AT24C256B</td><td>32 KB</td><td>A2</td><td>A1</td><td>A0</td><td>1/0</td><td>64</td><td>8</td></tr>
<tr><td>AT24C512B</td><td>64 KB</td><td>A2</td><td>A1</td><td>A0</td><td>1/0</td><td>128</td><td>8</td></tr>
</table>

注意：表 8－1 的片选引脚中，AT24C04 器件不用 A0 引脚，但要用 P0 位区分页地址，每页有 256 个字节(这里的“页”不要与页面写字节数中的“页”混淆)，在主机发出的寻址字节中，使 P0 位为 0 或 1，就可以访问 AT24C04 的 512 B 的内容；器件 AT24C08 和 AT24C16 的情况与此类似。

3. 主机写数据操作命令

1）写单字节

对 AT24C04 写入时，单片机发出起始信号“S”后接着发送的是器件寻址写操作(即

1010(A2)(A1)(P0)0，方向位为“0”)，然后释放 SDA 线并在 SCL 线上产生第 9 个时钟信号；被选中的 AT24C04 在 SDA 线上产生一个应答信号“A”；单片机再发送要写入的片内单元地址；收到 AT24C04 应答“0”后单片机发送数据字节，AT24C04 返回应答；然后单片机发出停止信号“P”，AT24C04 启动片内擦写过程。写入单字节的传输时序如图 8－8 所示。

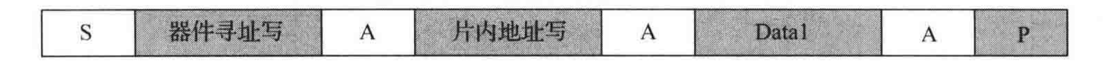


图 8－8　写入单字节的传输时序

2) 写多字节

要写入多个字节，可以利用 AT24C04 的页写入模式。AT24C04 的页为 16 字节。与字节写相似，首先单片机分别完成起始信号“S”操作、器件寻址写操作及片内单元首地址写操作，收到 AT24C04 应答“0”后单片机就逐个发送各数据字节，但每发送一个字节后都要等待应答。如果没有数据要发送了，单片机就发出停止信号“P”，AT24C04 就启动内部擦写周期，完成数据写入工作(约 10 ms)。

AT24C04 片内地址指针在接收到每一个数据字节后都自动加 1，在芯片的“页面写字节数”(16 字节)限度内，只需输入首地址。传送数据的字节数超过芯片的“页面写字节数”时，地址将“上卷”，前面的数据将被覆盖。写入 n 个字节的传输时序如图 8－9 所示。

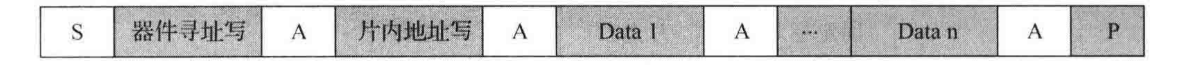


图 8－9　写入 n 个字节的传输时序

4. 主机读数据操作命令

1) 当前地址读

从 AT24C04 读数据时，单片机发出起始信号“S”后接着要完成器件寻址读操作，在第 9 个脉冲等待从机应答；被选中的从机在 SDA 线上产生一个应答信号“A”，并向 SDA 线发送数据字节；单片机发出应答信号和停止信号“P”。当前地址读传输时序如图 8－10 所示。

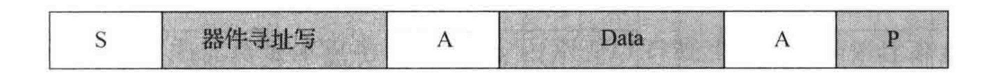
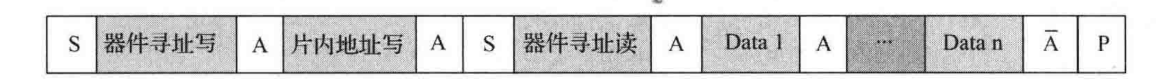


图 8－10　当前地址读传输时序

2) 随机读

随机读时，单片机也要先完成该器件寻址写操作和数据地址写操作(属于“伪写”，即方向控制位仍然为“0”)，均在第 9 个脉冲处等待从机应答。被选中的从机在 SDA 线上产生一个应答信号“A”。

收到器件应答后，单片机要先重复一次起始信号“S”并完成器件寻址读操作(即 1010(A2)(A1)(P0)1，方向位为“1”)，收到器件应答后就可以读出数据字节，每读出一个字节，单片机都要回复应答信号“A”。当最后一个字节数据读完后，单片机应返回非应答信号“$\overline{A}$”(高电平)，并发出停止信号“P”。随机读时序如图 8－11 所示。



S	器件寻址写	A	片内地址写	A	S	器件寻址读	A	Data 1	A	…	Data n	$\bar{A}$	P

图 8-11 随机读时序

5. AT24C04 的应用

【例 8-2】 扩展串行程序存储器 AT24C04 的系统连接原理图如图 8-7 所示，晶振是 11.0592 MHz，试编程实现向 AT24C04 中写入 4 个字符“SCMC”，地址不限。

分析： 题中 E^2PROM 选用 Atmel 公司的 AT24C04 芯片，用 AT89C51 单片机的通用 I/O 口与之相连。由于 AT89C51 没有 I^2C 接口，需要用软件模拟 I^2C 数据传输时序，这里用 P2.0 模拟数据线 SDA，用 P2.1 模拟时钟线 SCL。读写时，设定时钟线 SCL 的周期为 10 μs，由子程序 DEL5US 调用两次后，翻转电平。

有 4 个字符，AT24C04 每次能连续写入 16 个字节，我们需要一次即可写入，从起始地址开始写。

参考程序：

```
;定义变量
ACK     BIT     20H.0          ;应答标志位
SLA     EQU     30H            ;器件从地址变量
SUBA    EQU     31H            ;器件子地址变量
;定义常量
SDA     EQU     P2.0           ;数据线
SCL     EQU     P2.1           ;地址线
        ORG     0000H
        LJMP    MAIN
MAIN:   LCALL   DELAY          ;I²C 总线 AT24C04 上电延时
        MOV     SLA, #60H      ;器件从地址
        MOV     SUBA, #00H     ;器件子地址
MA10:   LCALL   STOP
MA20:   LCALL   START          ;启动总线
        MOV     A, SLA         ;指定从地址
        LCALL   WRBYTE         ;发送器件从地址
        LCALL   CACK           ;每发送一个字节调用 CACK 子程序，置应答位
        JNB     ACK, MA10      ;无应答则重发
        MOV     A, SUBA        ;指定子地址
        LCALL   WRBYTE         ;发送器件子地址
        LCALL   CACK           ;每发送一个字节调用 CACK 子程序，置应答位
        JNB     ACK, MA10      ;无应答则重发
        MOV     A, 'S'         ;取字符“S”
        LCALL   WRBYTE         ;开始写入数据
        LCALL   CACK           ;每发送一个字节调用 CACK 子程序，置应答位
        JNB     ACK, MA10      ;无应答则重发
```

```
        MOV     A, 'C'          ；取字符“C”
        LCALL   WRBYTE          ；开始写入数据
        LCALL   CACK            ；每发送一个字节调用 CACK 子程序，置应答位
        JNB     ACK, MA10       ；无应答则重发
        MOV     A, 'M'          ；取字符“M”
        LCALL   WRBYTE          ；开始写入数据
        LCALL   CACK            ；每发送一个字节调用 CACK 子程序，置应答位
        JNB     ACK, MA10       ；无应答则重发
        MOV     A, 'C'          ；取字符“C”
        LCALL   WRBYTE          ；开始写入数据
        LCALL   CACK            ；每发送一个字节调用 CACK 子程序，置应答位
        JNB     ACK, MA10       ；无应答则重发
        LCALL   STOP
        LJMP    $
；延时子程序，12 MHz，125 ms
DELAY:  MOV     R6, #00H
DE10:   MOV     R5, #00H
        DJNZ    R5, $
        DJNZ    R6, DE10
        RET
；延时子程序，12 MHz，5 μs(保证一个停止信号和起始信号的空闲时间大于 4.7 μs)
DEL5US: NOP
        NOP
        NOP
        NOP
        RET
；启动 I²C 总线子程序
START:  SETB    SDA
        NOP
        NOP
        SETB    SCL             ；起始条件建立时间大于 4.7 μs
        LCALL   DEL5US
        CLR     SDA             ；起始条件锁定时间大于 4 μs
        LCALL   DEL5US
        CLR     SCL             ；钳住总线，准备发数据
        NOP
        NOP
        RET
；结束 I²C 总线子程序
STOP:   CLR     SDA
        NOP
        NOP
```

```
        SETB    SCL         ；结束总线时间大于 4 μs
        LCALL   DEL5US
        SETB    SDA         ；结束总线
        LCALL   DEL5US
        RET
；检查应答位子程序(返回值，ACK=1 表示有应答)
CACK：  CLR     ACK         ；应答位初始清"0"
        SETB    SDA
        NOP
        NOP
        MOV     C，SDA
        JC      CEND        ；判断应答位
        SETB    ACK         ；应答位置"1"
CEND：  SETB    SCL         ；时钟电平"高"
        LCALL   DEL5US      ；延时 10 μs
        LCALL   DEL5US
        CLR     SCL         ；时钟电平"低"
        NOP
        NOP
        RET
；发送字节子程序(数据放入 ACC，时钟周期 10 μs)
WRBYTE：MOV     R0，＃08H   ；一个字节 8 位
WR10：  RLC     A           ；顺序发送数据位，高位在前，左移
        JNC     WR20        ；判断数据位(C=0，发"0"；C=1，发"1")
        SETB    SDA         ；发送"1"
        LCALL   DEL5US      ；延时 5 μs
        SETB    SCL
        LCALL   DEL5US
        LCALL   DEL5US
        CLR     SCL
        LCALL   DEL5US
        LJMP    WR30
WR20：  CLR     SDA         ；发送"0"
        LCALL   DEL5US
        SETB    SCL
        LCALL   DEL5US
        LCALL   DEL5US
        CLR     SCL
        LCALL   DEL5US
WR30：  DJNZ    R0，WR10    ；8 位未发完，重复
        RET
        END
```

8.3　SPI 总线接口及其扩展

SPI(Serial Peripheral Interface)总线是 Motorola 公司(摩托罗拉公司，其半导体器件部门独立后，更名为飞思卡尔半导体公司)推出的高速、全双工、同步串行通信总线。SPI 总线允许 MCU(微控制器)与各种外围设备以串行方式进行同步通信，属于全双工通信总线。SPI 总线广泛用于 E^2PROM、实时时钟、A/D 转换器、D/A 转换器等器件。

8.3.1　SPI 总线的基础知识

1. SPI 总线概述

SPI 总线通常有 3 根线：串行时钟线(SCK)、主机输入/从机输出数据线(MISO)和主机输出/从机输入数据线(MOSI)。除了上述 3 根线以外，一般还会有 1 根低电平有效的片选线($\overline{CS}$)，可以在多器件接入时使用。

SPI 工作模式有两种：主模式和从模式。SPI 允许一个主机启动一个从机进行同步通信，从而完成数据的同步交换和传输。只要主机有 SPI 控制器(也可用模拟方式)，就可以与基于 SPI 的各种芯片传输数据。

SPI 的串行总线通信协议是：由 SCK 提供时钟脉冲，MISO、MOSI 则基于此脉冲完成数据传输。主机数据输出时，MOSI 上的数据在时钟上升沿或下降沿时改变，在紧接着的下降沿或上升沿被从机读取，完成一位数据传输，输入也是同样原理。这样，在经历了 8 次时钟信号的改变(上升沿和下降沿为 1 次)后，就可以完成 8 位数据的传输。

要注意的是，SCK 信号线只由主机控制，从机不能控制。这样的传输方式与普通的串行通信(见 6.3 节)不同，普通的串行通信一次连续传输至少 8 位数据，而 SPI 允许数据一位一位传输。主机通过对 SCK 时钟线的控制可以完成对通信的控制，当没有时钟跳变时，从机不采集或传输数据。SPI 总线的输入/输出线分开，可同时进行数据传输，为全双工的通信方式。

不同的 SPI 设备，数据的改变和采集在时钟信号上升沿或下降沿有不同定义，将各种不同 SPI 接口片连到 MCU 的 SPI 总线时，应特别注意这些串行 I/O 芯片的输入/输出特性。

基本型 80C51 和 AT89C51 单片机没有配置 SPI 总线接口，但是可以利用其并行口线模拟 SPI 总线的时序，从而广泛地利用 SPI 接口的芯片资源。

2. SPI 总线的系统结构

MISO 和 MOSI 用于串行接收和发送数据，其数据的传输格式是高位(MSB)在前，低位(LSB)在后；SCK 是主机为从机提供同步时钟输入信号；$\overline{CS}$是片选使能信号。

SPI 总线的典型应用是单主机系统，该系统只有一台主机(单片机)，多个外围接口器件作为从机。单片机与多个 SPI 串行接口设备典型的 SPI 总线系统结构如图 8－12 所示。在这个系统中，只允许有一个作主机的 CPU 和若干具有 SPI 接口的外围器件(从机)。主机控制着数据向一个或多个从机的传输。从机只能在主机发命令时才能接收或向主机传输数据。所有的从机使用相同的时钟信号 SCK，并将所有从机的 MISO 引脚连接到主机的 MOSI 引脚，从机的 MOSI 引脚连接到主机的 MISO 引脚。但每个从机采用相互独立的片选信号来控制芯片使能端，使得在某一时刻只有一个从机有效。

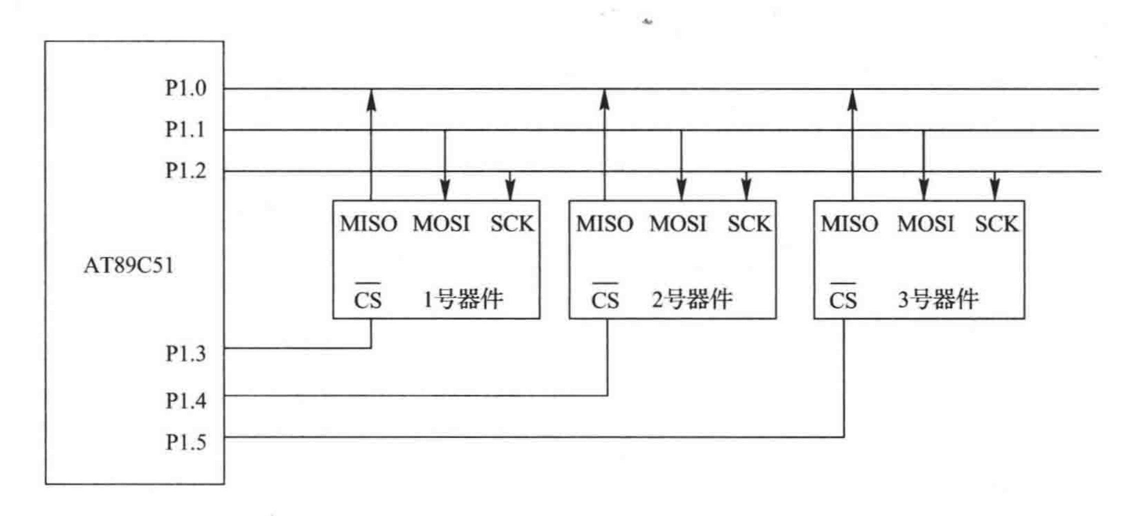


图 8-12 单片机扩展 SPI 的系统结构

当有多个不同的 SPI 器件连至 SPI 总线上作为从机时，必须注意两点：一是其必有片选端；二是其接 MISO 线的输出脚必须有三态，片选无效时输出高阻态，以不影响其他 SPI 设备的正常工作。

8.3.2 SPI 总线的数据传输时序

SPI 总线的数据传输过程中需要时钟驱动。SPI 总线的时钟信号 SCK 有时钟极性(CPOL)和时钟相位(CPHA)两个参数，前者决定有效时钟是高电平还是低电平，后者决定有效时钟的相位，这两个参数配合起来决定 SPI 总线的数据传输时序。

在片选信号$\overline{CS}$有效时，对数据传输线(MOSI 或 MISO)上的采样在 SCK 信号的上升沿或下降沿均可。如果采样跳变沿是 SCK 信号的第 1 个跳变沿，则相位控制位 CPHA 为 0；如果采样跳变沿是 SCK 信号的第 2 个跳变沿，则相位控制位 CPHA 为 1。SCK 空闲时有两种极性，低电平对应 CPOL 为 0，高电平对应 CPOL 为 1。

图 8-13 所示为 SPI 总线 4 种工作模式的时序图。从时序图可以看出，SPI 协议仅规

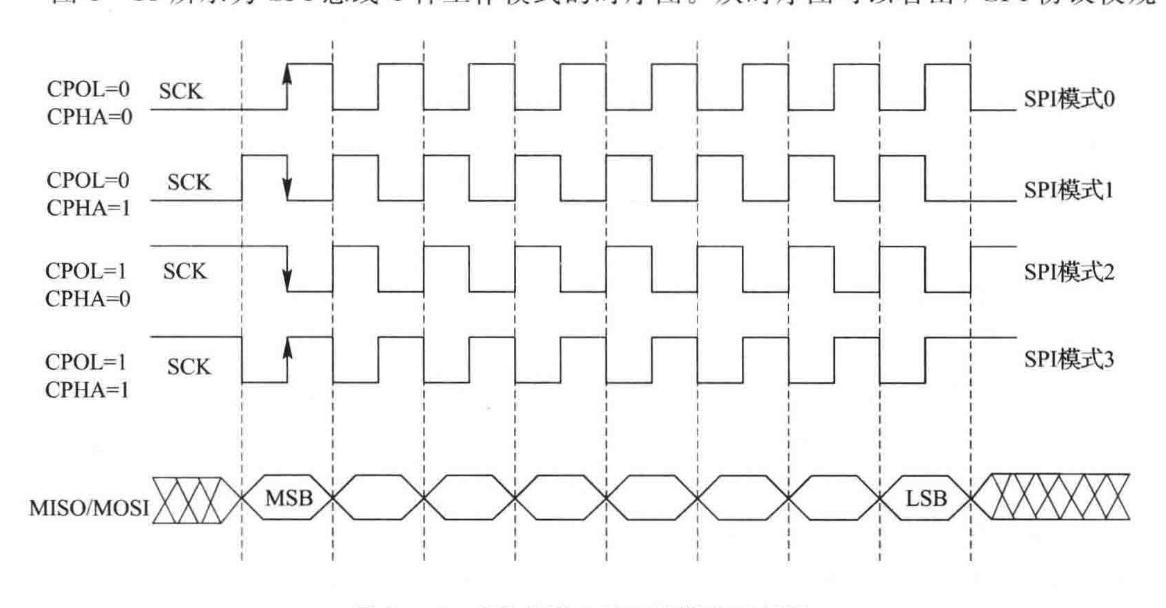


图 8-13 SPI 总线 4 种工作模式时序图

定了每一帧数据如何传输，并未规定帧结构的组成。CPOL 和 CPHA 两个参数决定了 SPI 的 4 种工作模式。CPOL 控制在没有数据传输时时钟的空闲状态电平为 0 或 1 状态，CPHA 控制数据采样的时钟是第 1 个跳变沿还是第 2 个跳变沿。

具有标准 SPI 接口的微控制器可以通过配置工作方式与相应的外设接口器件进行连接。对于没有标准 SPI 接口的 MCS－51 单片机，要想与 SPI 扩展器件传输数据，就要利用通用 I/O 口通过软件来模拟，这时必须严格依据器件的操作时序。

8.3.3 E²PROM 芯片 X25045

X25045 是一种集上电复位控制、电源监控、看门狗、定时器和 512×8 位 E²PROM 四种功能于一块芯片的多功能器件。

1. X25045 的引脚定义

X25045 引脚图如图 8－14 所示。

SO：串行输出线。

$\overline{\text{WP}}$：写保护输入端。

RESET：复位输出端。

SCK：同步时钟输入线。

SI：串行输入线。

$\overline{\text{CS}}$/WDI：片选输入/看门狗复位输入端。

V_{SS}：地。

V_{CC}：电源。

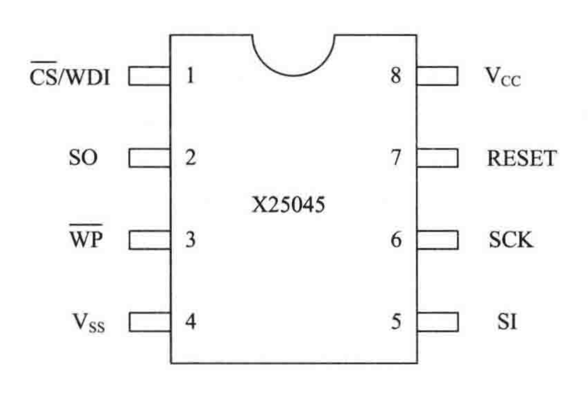


图 8－14　X25045 引脚图

2. X25045 的功能

X25045 芯片的功能如下：

(1) 上电复位控制。在对 X25045 通电时，RESET 引脚输出有效的复位信号，并保持至少 200 ms，使 CPU 有效复位。

(2) 电源电压监控。当检测到电源电压低于内部门槛电压 U_{trip} 时，RESET 输出复位信号，直至电源电压高于 U_{trip} 并保持至少 200 ms，复位信号才被撤销。U_{trip} 的出厂值根据芯片型号不同共有 5 个级别的电压范围。对于需要电源电压精确监控的应用，用户可以搭建编程电路，对芯片内 U_{trip} 电压进行微调。

(3) 看门狗定时器。芯片内部状态寄存器的WD1、WD0是看门狗定时设置位，通过状态寄存器写命令WRSR修改这两个标志位，就能选择3种定时间隔或关闭定时器。定时器溢出，产生RESET信号。$\overline{CS}$/WDI引脚上输入下降沿完成对看门狗定时器的复位。看门狗的定时值见表8-2。

表8-2 看门狗的定时值

WD1	WD0	看门狗定时值
0	0	1.4 s
0	1	600 ms
1	0	200 ms
1	1	关闭定时器

(4) 内含512B串行E^2PROM。可擦写10万次，数据保存时间为100年。该芯片设计了3种保护方式防止误写，使产生误写的可能性极小。

当$\overline{WP}$写保护引脚被拉低时，内部存储单元和状态寄存器都禁止写入。

通过对状态寄存器的BL1、BL0位的设置，可以选择对不同的存储区域进行写保护，具体见表8-3。

表8-3 X25045保护区域

BL1	BL0	写保护的区域
0	0	没有保护
0	1	180H～1FFH
1	0	100H～1FFH
1	1	000H～1FFH

3. X25045的操作命令

在进行任何写操作前都必须使用WREN命令，打开写使能开关，而在上电初始化或写操作完成时，写使能开关自动关闭。

X25045共有6条操作命令，见表8-4。

表8-4 X25045的操作命令

命令名称	指令格式	功　能
WREN	00000110(06H)	打开写使能开关
WRDI	00000100(04H)	关闭写使能开关
RDSR	00000101(05H)	读状态寄存器
WRSR	00000001(01H)	写状态寄存器
READ	0000(A8)011(03H或0BH)	读存储单元，A8为页地址
WRITE	0000(A8)010(02H或0AH)	写存储单元，A8为页地址

X25045的状态寄存器描述器件的当前状态，各位含义如下：

D7	D6	D5	D4	D3	D2	D1	D0
0	0	WD1	WD0	BL1	BL0	WEL	WIP

其中，WD1、WD0 是看门狗定时时间设置位；BL1、BL0 是存储单元写保护区设置位；WEL 是只读标志，1 表示写使能开关打开；WIP 也是只读标志，1 表示芯片内部正处于写周期。上电复位时，各位都被清零。

4. X25045 的工作时序

图 8－15 为 E^2PROM X25045 与 89C51 的接口电路。由于 89C51 内部没有 SPI 接口硬件，因此利用 4 根 I/O 线 P1.4、P1.5、P1.6、P1.7 作为 MISO、$\overline{WP}$、MOSI、SCK 的控制线。P1.3 接 X25045 的片选端$\overline{CS}$。

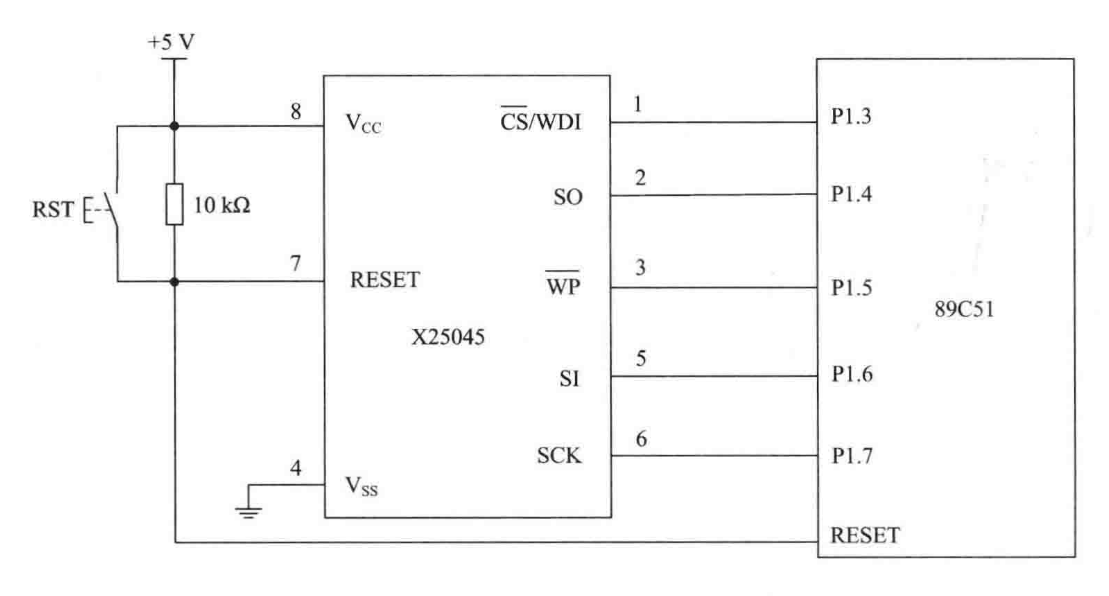


图 8－15　X25045 与 89C51 的接口电路

在对 X25045 进行读/写操作时，需先在 MOSI 线上输出 READ 或 WRITE 指令（指令码 A8 代表存储单元地址的最高位），接着输出低 8 位地址，即可连续读出或写入数据。其中，读指针和写指针的工作方式完全不同，读指针的全部 8 位用来计数，FFH 溢出后变成 00H；写指针只用最低两位计数，××××××11B 溢出后变成××××××00B，所以连续写的实际结果是在 4 个单元中反复写入。由于 E^2PROM 的写入时间长，所以在连续两条写指令之间应读取状态寄存器中的 WIP 状态，只有在内部写周期结束时才可输入下一条写指令。另外，因为上电初始化或写操作完成时写使能开关自动关闭，在进行任何写操作前都必须使用 WREN 命令，打开写使能开关。其字节读/写时序如图 8－16 和图 8－17 所示。

在对 X25045 内部的状态寄存器进行 RDSR 或 WRSR 操作时，在 MOSI 线输出指令 RDSR 最后一位后，紧跟着 MISO 线即输出状态寄存器内容，如图 8－18 所示，而 WRSR 操作需要在 SI 线上紧跟着 WRSR 命令之后输出 8 位状态数据。

5. X25045 的应用

【例 8－3】 X25045 与 89C51 的接口电路如图 8－15 所示，X25045 的 SPI 接口时序是

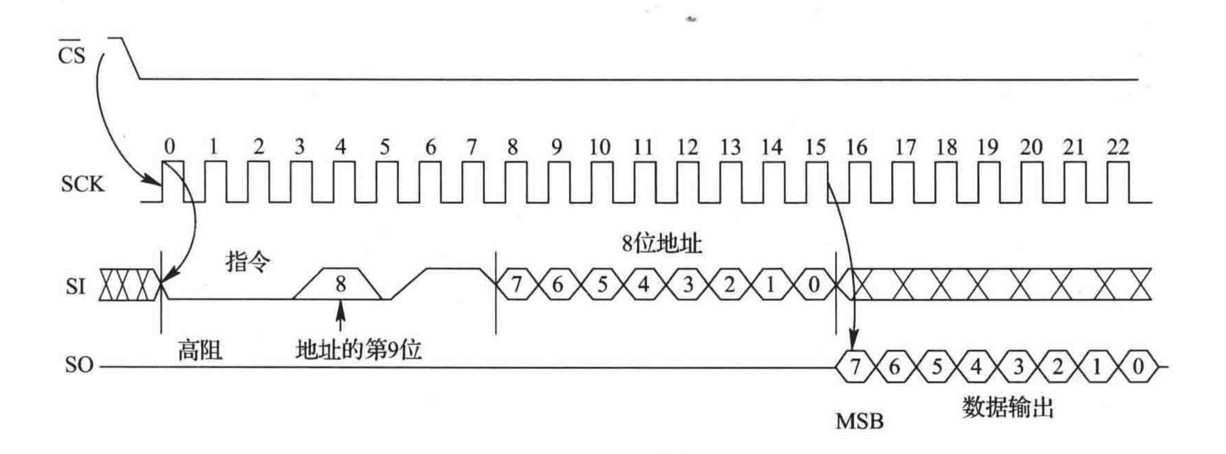


图 8－16　X25045 读字节时序

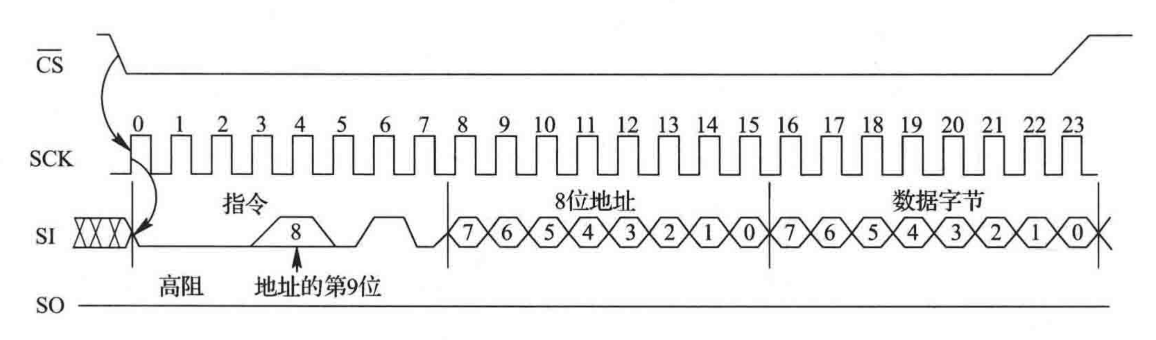


图 8－17　X25045 写字节时序

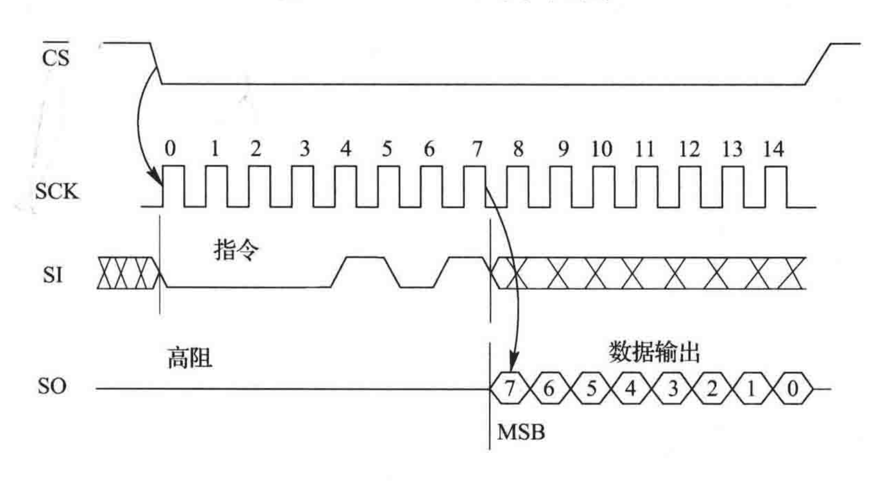


图 8－18　X25045 读状态时序

上升沿输入和下降沿输出，用软件来模拟 SPI 总线的读/写时序。要求从 X25045 的第 1 页的 10H 开始读出 32 个字节的数据，并写入到地址 30H 开始的内存单元中。

分析：用单片机的 P1.4 和 P1.6 来模拟 SPI 总线的输出和输入，用 P1.7 模拟时钟信号。先向 MOSI 输出读存储单元命令，将页地址“1”插入到读命令中；再将要读的地址输出到 MOSI 总线上。

参考程序：

```
CS      BIT     P1.3          ；片选信号
MISO    BIT     P1.4          ；X25045 串行数据输出
MOSI    BIT     P1.6          ；X25045 串行数据输入
WP      BIT     P1.5          ；X25045 工作开关，0 为禁止写入；1 为允许写入
SCK     BIT     P1.7          ；时钟
READ    EQU     03H           ；READ 指令
        ORG     0000H
        LJMP    MAIN
MAIN：  MOV     R0，#30H      ；内存首地址
        MOV     R2，#20H      ；32 个字节
        MOV     R3，#10H      ；要读出数据的 X25045 存储器首地址
        SETB    F0            ；页号为“1”
        SETB    WP            ；允许 X25045 写入数据
        LCALL   READN
        CLR     WP            ；禁止 X25045 写入数据
        LJMP    $
；把指定地址开始的 E²PROM 单元数据读出并放入 RAM 单元
；R0 是片内 RAM 单元缓冲区首地址，R2 是要读的字节长度，R3 是 X25045 的 8 位地址
READN：LCALL   STAX          ；启动 X25045
        MOV     A，READ       ；输出 READ 指令
        MOV     C，F0         ；页地址在 F0
        MOV     ACC.3，C      ；页地址写入 READ
        LCALL   OUTBYTE       ；调用写 X25045 子程序
        MOV     A，R3
        LCALL   OUTBYTE       ；调用写 X25045 子程序
READ1：LCALL   INBYTE        ；调用读 X25045 子程序
        MOV     @R0，A
        INC     R0
        DJNZ    R2，READ1
        LCALL   ENDX
        RET
STAX：  SETB    CS            ；启动 X25045 指令
        NOP                   ；先置高 CS，再置低 SCK，再拉低 CS
        CLR     SCK
        NOP
        CLR     CS
        NOP
        RET
ENDX：  CLR     SCK           ；结束 X25045 指令
```

```
           SETB   CS           ；先置低 SCK，再置高 CS
           NOP
           NOP
           RET
；向 X25045 E²PROM 写入 8 位地址或数据，高位在前，低位在后，内容在 A 中
OUTBYTE：  MOV    R7，#08H     ；置循环次数 8
OUTBYT1：  CLR    SCK
           RLC    A            ；ACC 的最高位送 Cy
           MOV    MOSI，C      ；Cy 送 X25045 的 SI
           SETB   SCK
           DJNZ   R7，OUTBYT1  ；循环 8 次
           CLR    SCK
           RET
；从 X25045 E²PROM 中读出 8 位数据，高位在前，低位在后
；A 的内容为读出的 8 位数据
INBYTE：   MOV    R7，#08H     ；置循环次数 8
INBYT1：   SETB   SCK
           CLR    SCK          ；SCK 的下降沿数据出现在 SO 端
           MOV    C，MISO      ；数据输出端的数据(1bit)送入 0
           RLC    A
           DJNZ   R7，INBYT1
           RET
           END
```

上述的接口电路和程序同样也适用于其他具有相同时序的 SPI 串行外围接口芯片，如 A/D 转换芯片、网络控制器芯片、LED 驱动芯片等。

8.3.4 A/D 转换器芯片 TLC549

TLC549 是 TI 公司生产的一种低价位、高性能的 8 位 A/D 转换器，以 8 位开关电容逐次逼近的方法实现 A/D 转换。TLC549 的转换时间小于 17 μs，最大转换速率为 40 kHz，工作电压为 3～6 V。TLC549 可以采用 SPI 总线方式与单片机进行接口。

1. TLC549 的引脚定义

TLC549 引脚图如图 8－19 所示。

$V_{REF(+)}$：正基准电压端，2.5 V$\leqslant V_{REF(+)} \leqslant V_{CC}+0.1$。

$V_{REF(-)}$：负基准电压端，-0.1 V$\leqslant V_{REF(-)} \leqslant 2.5$ V，且要求：$V_{REF(+)}-V_{REF(-)} \geqslant 1$ V。

$\overline{CS}$：芯片选择输入端，要求输入高电平 $V_{IN} \geqslant 2$ V，输入低电平 $V_{IN} \leqslant 0.8$ V。

DATA OUT：转换结果数据串行输出端，与 TTL 电平兼容，输出时高位在前，低位在后。

ANALOG IN：模拟信号输入端，$0 \leqslant V_{ANALOG\ IN} \leqslant V_{CC}$。当 $V_{ANALOG\ IN} \geqslant V_{REF(+)}$ 时，转换结果为全“1”(0FFH)；当 $V_{ANALOG\ IN} \leqslant V_{REF(-)}$ 时，转换结果为全“0”(00H)。

I/O CLOCK：外接输入/输出时钟输入端，用于同步芯片的输入/输出操作，无需与芯片内部系统时钟同步。

V_{CC}：系统电源，3 V$\leqslant V_{CC}\leqslant$6 V。

GND：接地端。

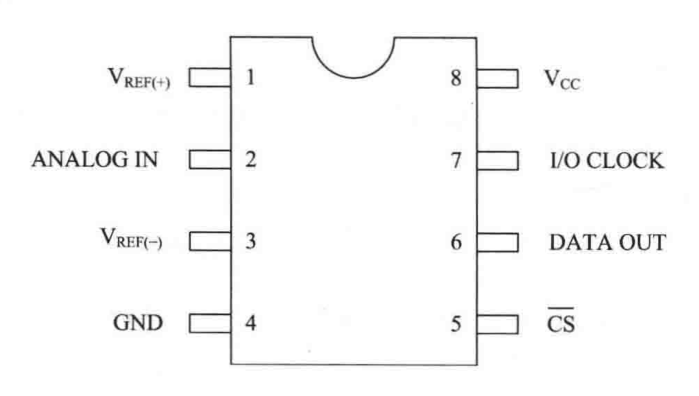


图 8-19　TLC549 引脚图

2. TLC549 的功能框图

TLC549 由采样保持器、模/数转换器、输出数据寄存器、数据选择与驱动器及相关控制逻辑电路组成。TLC549 的内部结构如图 8-20 所示。

TLC549 带有片内系统时钟，该时钟与 I/O CLOCK 是独立工作的，无需特殊的速度及相位匹配。当$\overline{CS}$为高时，数据输出端 DATA OUT 处于高阻状态，此时 I/O CLOCK 不起作用。这种$\overline{CS}$控制作用允许在同时使用多片 TLC549 时，共用 I/O CLOCK，以减少多片 A/D 使用时的 I/O 控制端口。

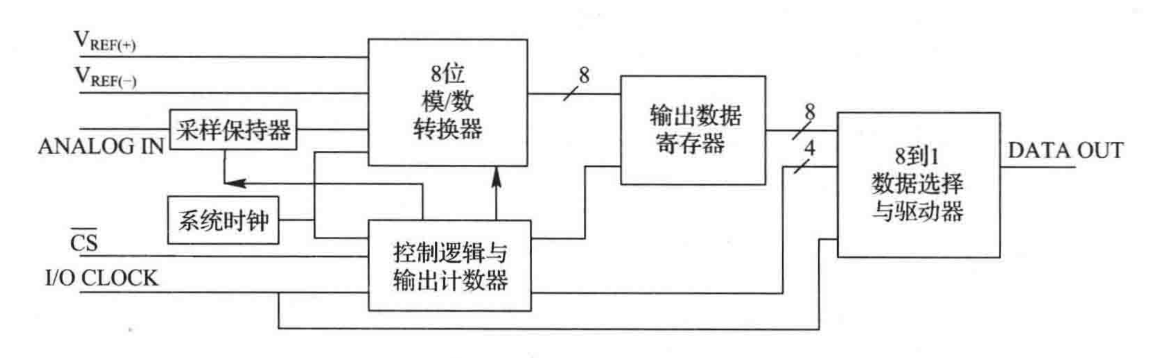


图 8-20　TLC549 的内部结构

3. TLC549 的工作时序

TLC549 的工作时序如图 8-21 所示。

（1）$\overline{CS}$置低电平，内部电路测得$\overline{CS}$下降沿后，在等待两个内部时钟上升沿和一个下降沿后，再确认这一变化，最后自动将前一次转换结果的最高位 D7 输出到 DATA OUT。

（2）在前 4 个 I/O CLOCK 周期的下降沿依次移出 D6、D5、D4、D3，片上采样保持电路在第 4 个 I/O CLOCK 下降沿开始采样模拟输入。

（3）接下来的 3 个 I/O CLOCK 周期的下降沿可移出 D2、D1、D0 各位。

（4）在第 8 个 I/O CLCOK 后，$\overline{CS}$必须为高电平或 I/O CLOCK 保持低电平，这种状态需要维持 36 个内部系统时钟周期以等待保持和转换工作的完成。

应注意，此时的输出是前一次的转换结果而不是正在进行的转换结果。若要在特定的时刻采样模拟信号，则应使第 8 个 I/O CLOCK 时钟的下降沿与该时刻对应。因为芯片虽

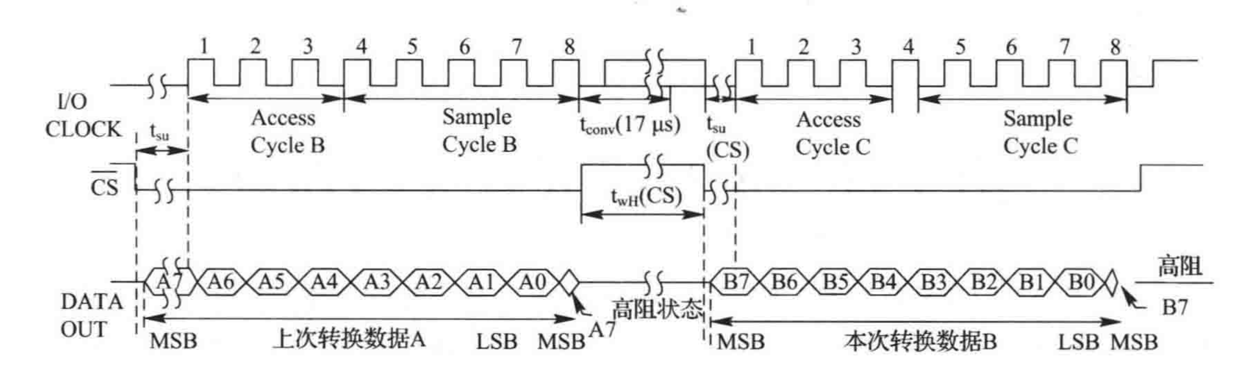


图 8-21　TLC549 的工作时序

在第 4 个 I/O CLOCK 时钟的下降沿开始采样，却在第 8 个 I/O CLOCK 的下降沿才开始保存。

思考与练习

1. I^2C 总线的特点是什么？

2. I^2C 总线的起始信号和停止信号是如何定义的？

3. I^2C 总线的数据传送方向如何控制？

4. 具备 I^2C 总线接口的 E^2PROM 芯片有哪几种型号？容量如何？

5. 简述 AT24CXX 系列芯片的读写格式。

6. SPI 接口总线有哪几个？作用是什么？

7. 简述 SPI 数据传输的基本过程。

8. AT89C51 的 P3.0、P3.1 分别通过数据线 SDA、时钟线 SCL 与 AT24C04 相连，晶振频率为 12 MHz，试编写程序将 AT24C04 首地址开始的 16 个字节数据写入单片机内存 30H～3FH 中。

第9章　单片机开发入门知识

本章主要介绍单片机开发工具和Keil μVision4集成开发环境两部分内容，重点讲述Keil μVision4集成开发环境中源程序的建立、编译、调试等内容。由于篇幅所限，本章仅介绍两个实际应用开发案例，有兴趣的读者可以登录www.mcs-51.com网站学习其他实际案例。

9.1　单片机应用系统开发技术

由于自身软/硬件的限制，单片机本身并无开发能力，必须借助开发工具来开发应用软件以及对硬件系统进行诊断。由于单片机应用系统一般要进行系统硬件的扩展与配置，同时还需要开发相应的软件，因此，开发者研制一个较完整的单片机产品时，必须完成以下工作：

（1）硬件电路设计、制板、组装。

（2）应用软件的编写、调试。

（3）应用系统的程序固化、脱机（脱离开发系统）运行。

9.1.1　单片机应用系统的开发过程

单片机应用系统的开发过程包括系统方案论证、系统硬件设计、系统软件设计、系统仿真调试和脱机运行调试。各部分的详细内容如图9-1所示。

1. 系统方案论证和总体设计

方案讨论包括查找资料，分析研究，并解决以下问题：

（1）了解国内类似系统的开发水平、供应状态；如果是委托研制项目，还应充分了解系统的技术要求、应用环境，以确定项目的技术难度。

（2）了解可移植的软件、硬件技术。能够移植的尽量移植，防止大量的低水平重复劳动。

（3）摸清软、硬件技术的关键，明确技术主攻方向。

（4）综合考虑软、硬件分工与配合。单片机应用系统设计中，软、硬件工作有密切的相关性。

（5）通过调查研究，确定应用系统的功能和技术指标，软、硬件技术方案及分工。

从总体上来看，设计任务可分为硬件设计和软件设计，这两者互相结合，不可分离。从时间上来看，系统的硬件设计与软件设计可同时进行。硬件设计的绝大部分工作量是在最初阶段，到后期往往还要做一些修改。只要技术准备充分，硬件设计的大返工是较少的。软件设计的任务贯彻始终，到中后期基本上都是软件设计任务。随着集成电路技术的飞速发展，各种功能很强的芯片不断出现，与软件相关的硬件电路的设计就变得越来越简

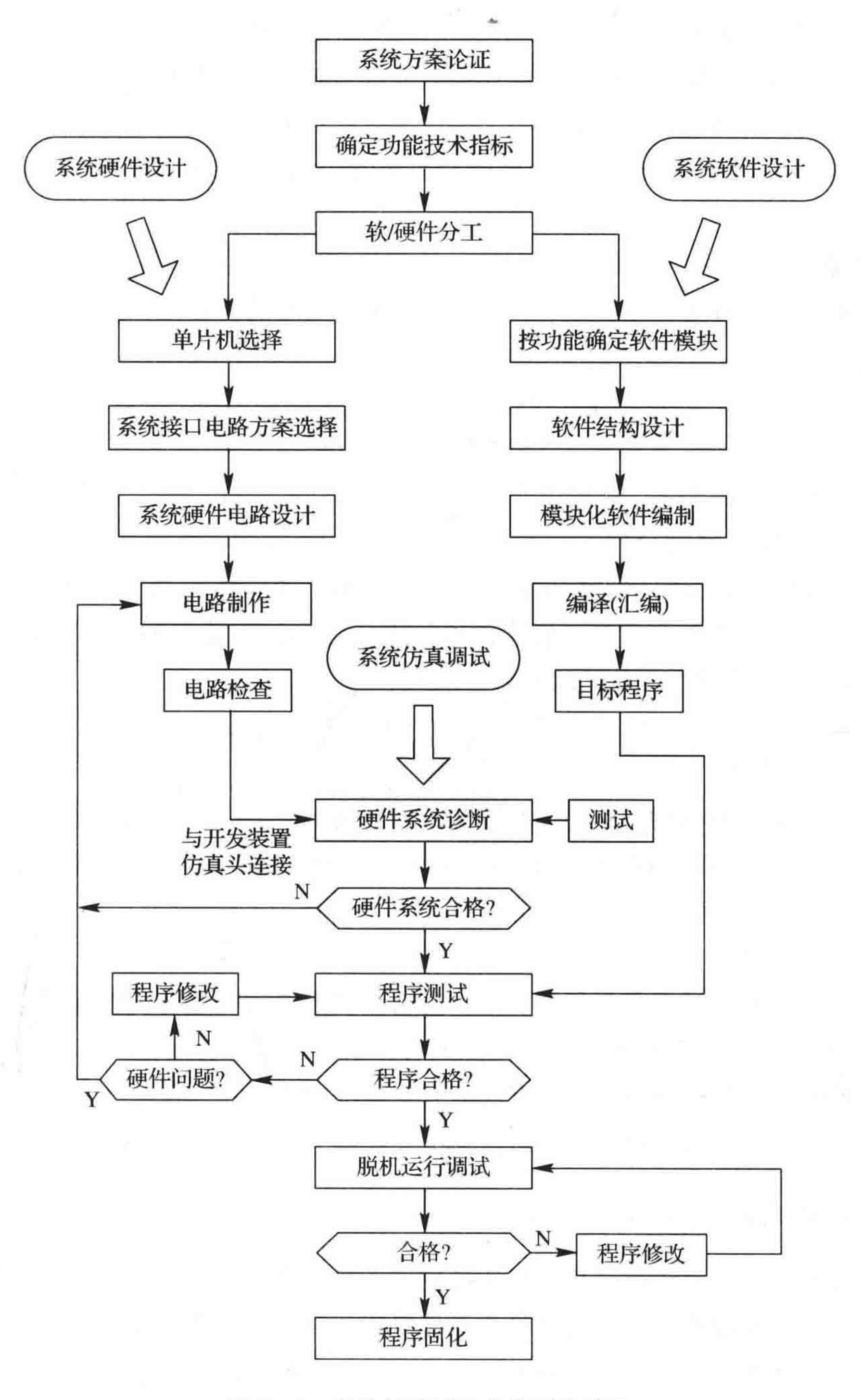


图 9-1 单片机应用系统的开发流程

单，在整个项目中占的比重逐渐减轻。

2. 系统硬件设计

硬件设计就是在总体方案的指导下，对构成单片机应用系统的所有功能模块进行详细、具体的电路设计，包括：具体确定系统中所要使用的元器件，设计出系统的电路原理图，必要时做一些部件实验，以验证电路的正确性；进行工艺结构的设计加工、印制电路板的制作及样机的组装等。

单片机应用系统的设计可划分为两部分：一部分是与单片机直接接口的数字电路芯片的选择和设计，如存储器和并行接口的扩展，定时系统、中断系统的扩展，一般外部设备的接口，与 A/D、D/A 芯片的接口；另一部分是与模拟电路相关的电路设计，包括模拟信号的采集、整形、放大、变换、隔离和传感器的选用，输出通道的隔离和驱动及执行元件的选用。

3. 系统软件设计

软件设计是根据任务要求并结合硬件设计，采用熟悉的语言(汇编语言或 C 语言)完成程序的设计。

单片机应用系统是一个整体，当系统的硬件电路设计定型后，软件的任务也就明确了。设计单片机系统应用软件时，应注意以下几个方面：

(1) 根据软件功能要求，将软件分成若干相对独立的部分，设计出合理的软件总体结构，使其清晰、简洁、流程合理。

(2) 功能程序实行模块化、子程序化，既便于调试、链接，又便于移植和修改。

(3) 对于复杂的模块和系统，应绘制出程序流程图。多花一些时间来设计程序流程图,可以大大减少源程序编写、调试的时间。

(4) 在程序的相关位置处写上功能注释，可提高程序的可读性。

4. 系统仿真调试

调试是一个非常复杂的过程，一般情况下需要借助开发工具(开发系统)，通过运行程序来观察开发的单片机应用系统(目标板)是否符合设计要求。在确保硬件电路设计正确、合理的前提下，调试过程实质上是程序反复修改的过程。

5. 脱机运行调试

软件和硬件联机调试反复运行正常后，借助开发系统的编程器，将程序“写入”单片机应用系统的程序存储器 EPROM 或 E^2PROM 中，这个过程称为固化。

固化完成后，单片机应用系统即可脱离开发系统独立工作。这时还需将单片机应用系统带到现场投入实际工作，检验其可靠性和抗干扰能力，直到完全满足要求。

9.1.2 单片机开发调试工具

从硬件电路设计、源程序编写到单片机应用系统正常工作前的全过程，统称为单片机应用系统的开发，而辅助这一开发过程的工具便称为开发工具或开发系统。开发系统本身也是一个计算机系统，在完成上述开发任务时，可进行仿真。仿真是把应用系统自身的单片机拔掉，将开发系统的仿真插头插入，以取代原单片机，从而实现对应用样机软、硬件的故障诊断和调试。

开发工具应具备以下主要作用：

(1) 系统硬件电路的诊断。

(2) 源程序的输入与修改。

(3) 除连续运行程序外，具有单步运行、设断点运行和状态查询等功能。

(4) 能将程序固化到 EPROM 芯片上去。

单片机开发工具有很多，包括集成开发平台（如 Keil μVision），系统仿真工具（如 Proteus 仿真软件），用于电路原理图设计和电路板布线的工具（如 Protel、Altium Designer 等软件），用于目标板调试的各种型号仿真器，用于固化目标代码的各种编程器以及用于数字电路时序分析的示波器、逻辑分析仪和逻辑笔等。

1. 仿真器

仿真器（Emulator）具有以某一系统复现另一系统的功能，它与计算机软件模拟（详见9.2.3小节）的区别在于，仿真器用于模拟单片机系统的外在表现、行为，而不是模拟单片机系统的抽象模型。某种型号的仿真器如图 9-2 所示。

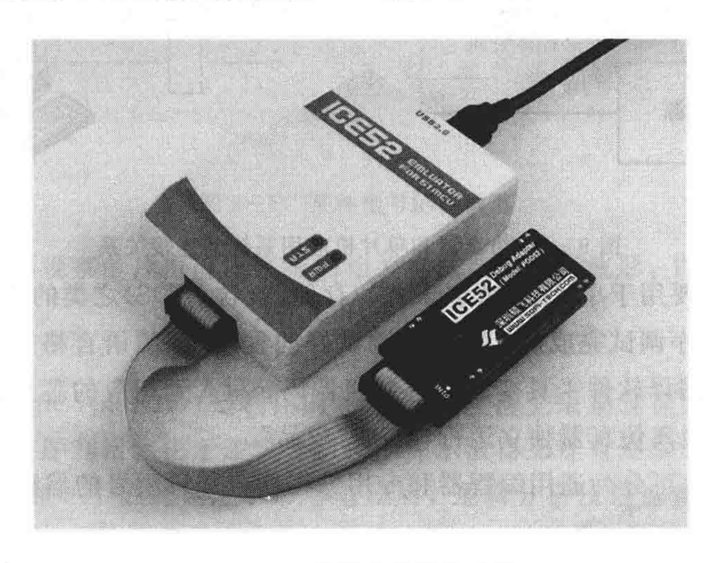


图 9-2　某种型号的仿真器

仿真器是用以实现硬件仿真的工具。仿真器可以替代单片机对程序的运行进行控制，如单步、全速、查看资源、设置断点等。尽管软件仿真具有无须搭建硬件电路就可以对程序进行验证的优点，但软件仿真无法完全反映真实硬件的运行状况，因此还要通过硬件仿真来完成最终的设计。目前，开发过程中硬件仿真是必需的。

仿真，就是用开发系统的资源来仿真应用系统，此时开发系统便是仿真器。一般多采用在线仿真，即仿真器控制的硬件环境与应用系统完全一致，或就是实际的应用系统。

仿真方法是：拔下应用系统（用户板）的 CPU，改插开发系统的仿真头，两个系统便共用一个 CPU，而仿真器的存储器中可以存放应用系统的程序。仿真器运行该程序，就可以测试应用系统的硬件功能和软件功能。这就是所谓“出借”CPU 的方法。仿真器可以连续运行程序、单步运行程序或设断点运行，也可以进行状态查询等。

仿真器除了具有如图 9-2 所示的硬件部分以外，一般还配有在微机上运行的专门软件程序（一般由仿真器生产厂家提供），两者共同组成仿真系统。所有的仿真操作命令都是在软件程序上操作的。大部分仿真器还可以和 Keil μVision 集成开发环境一起使用。

图 9-3 是仿真器和单片机应用系统（用户板）的连接关系图。

2. 编程器

编程器（Programmer）也称烧录器，是一个给可编程集成电路芯片写上数据（二进制程

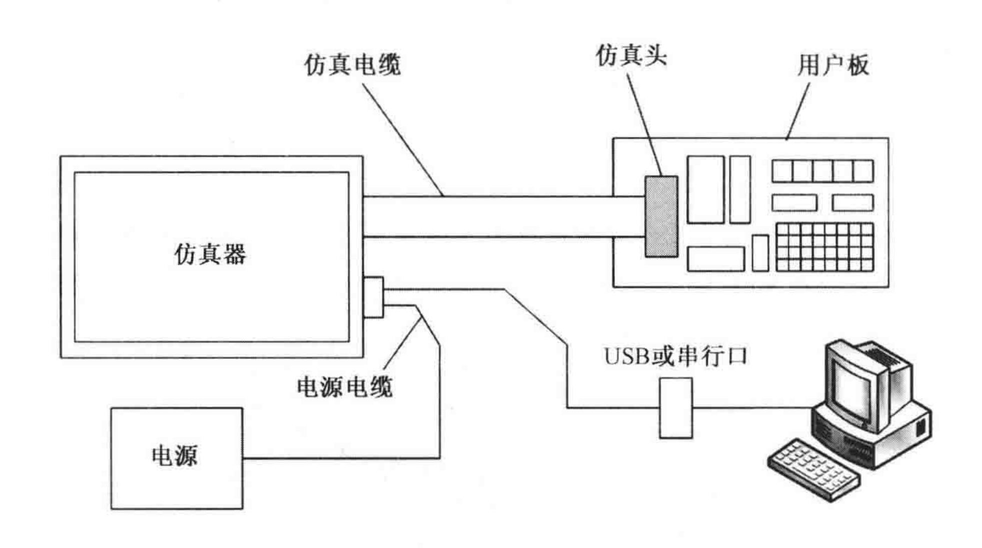


图 9-3 仿真器和单片机应用系统的连接关系

序代码)的工具，主要用于单片机(含嵌入式)/存储器(含 BIOS)之类的芯片的编程(或称为刷写、固化)。当程序调试完成后，需要将调试好的程序(汇编语言格式或 C 语言格式)通过汇编软件工具或编译软件工具变成二进制机器码，写入到相应的芯片中，使得开发的单片机应用系统可以脱离仿真器独立运行，变成“成品”。

编程器在功能上可分为通用编程器和专用编程器。某种型号的编程器如图 9-4 所示。



图 9-4 某种型号的编程器

目前，某些型号的单片机支持 ISP 功能，如 STC 系列。例如，通过 STC-ISP 单片机串口编程烧录软件直接把程序烧录到芯片中，而无需使用编程器。

3. 示波器

示波器(Oscilloscope)是一种用途十分广泛的电子测量仪器，如图 9-5 所示。它能把肉眼看不见的电信号变换成看得见的图像，便于人们研究各种电现象的变化过程。利用示波器能观察各种不同信号幅度随时间变化的波形曲线，还可以用它测试各种不同的电量，如电压、电流、频率、相位差、调幅度等。

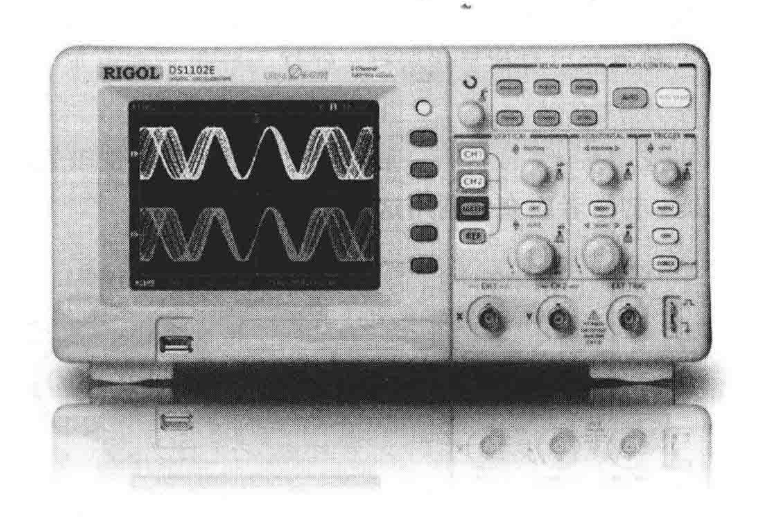


图9-5 某种型号的示波器

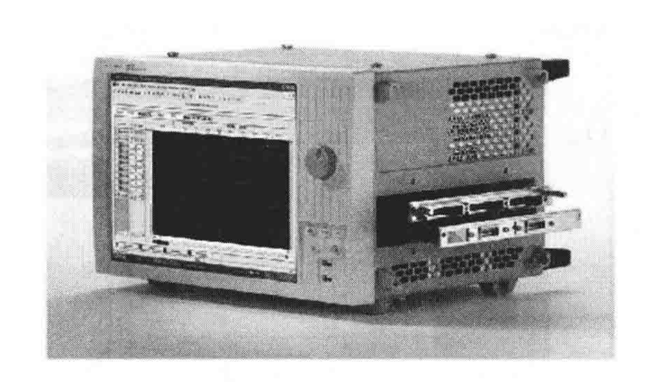
可以通过示波器观察单片机应用系统的相关测试点的电压波形，以此判断单片机系统工作是否正常。

4. 逻辑分析仪

逻辑分析仪(Logic Analyzer)是利用时钟从测试设备上采集和显示数字信号的仪器，最主要的作用在于时序判定。由于逻辑分析仪不像示波器那样有许多电压等级，通常只显示两个电压(逻辑1和0)，因此设定了参考电压后，逻辑分析仪根据被测信号通过比较器后的比较结果进行判定，被测信号电压高于参考电压者为High，低于参考电压者为Low，在High与Low之间形成数字波形。

逻辑分析仪主要用于复杂数字电路(单片机应用系统)的调试，可以检查多路时序之间的关系，这种定时分析可以对输入数据进行有效采样，跟踪采样时产生的任何跳变，从而容易识别毛刺(毛刺是采样时多次穿越逻辑阈值的跳变，难以查找)。

逻辑分析仪主要有两种，一种是独立的，另一种是将一块板卡插入计算机插槽中和计算机配合使用，如图9-6所示。



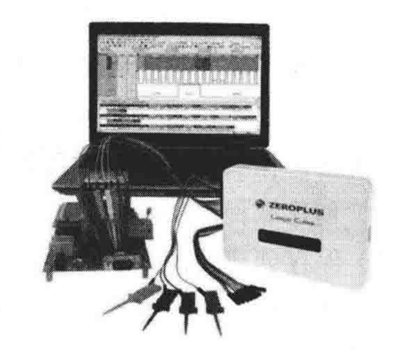


图9-6 两种类型的逻辑分析仪

5. 逻辑笔

逻辑笔(Logic Test Pen)是采用不同颜色的指示灯来表示数字电平高低的仪器，如图

9－7所示。它是测量数字电路简便易用的工具。使用逻辑笔可快速测量出数字电路中有故障的芯片。逻辑笔上一般有二三只信号指示灯，红灯一般表示高电平，绿灯一般表示低电平，黄灯表示所测信号为脉冲信号。

对于简单的单片机应用系统，或进行一般的电平判断，采用逻辑笔比较好。而且逻辑笔价格便宜、使用便捷，初学者应充分利用这一工具。

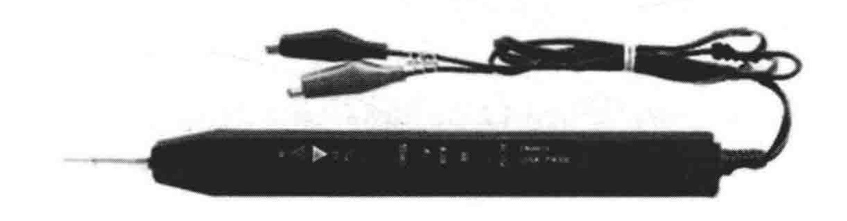


图 9－7　逻辑笔

9.2　Keil 集成开发平台

Keil 软件是 Keil Software 公司出品的开发 MCS－51 系列单片机应用系统的开发平台。Keil 软件是目前流行的开发 MCS－51 系列单片机的软件，提供了包括源程序编辑器、C 编译器、汇编器、链接器、库管理和一个功能强大的仿真调试器等在内的开发工具，并通过一个集成开发环境(Keil μVision)将这些部分组合在一起。本书以 μVision4 英文版本为例进行讲解(μVision4 有汉化版本，可以参照学习)。掌握这一软件的使用方法对于 MCS－51 系列单片机应用系统的开发者来说是十分重要的，如果使用 C 语言编程，那么 Keil 几乎是开发者的不二之选，即使不使用 C 语言而仅用汇编语言编程，其方便易用的集成环境、强大的软件仿真调试工具也会使开发者达到事半功倍的效果。

9.2.1　应用程序的创建

应用程序的创建过程大致如下：

(1) 新建一个工程项目文件。

(2) 为工程选择目标器件(如 AT89C51)。

(3) 为工程项目设置软硬件调试环境。

(4) 创建源程序文件。

(5) 保存创建的源程序文件。

(6) 把源程序文件添加到项目中。

首先启动 Keil μVision4。从计算机桌面上直接双击 Keil μVision4 图标即可启动该软件。Keil μVision4 提供一个菜单栏、一个工具栏，以便快速选择命令按钮。另外，Keill μVision4 还有文本编辑窗口、输出信息窗口、工程窗口，如图 9－8 所示。Keil μVision4 允许同时打开多个编辑窗口，浏览多个源文件。

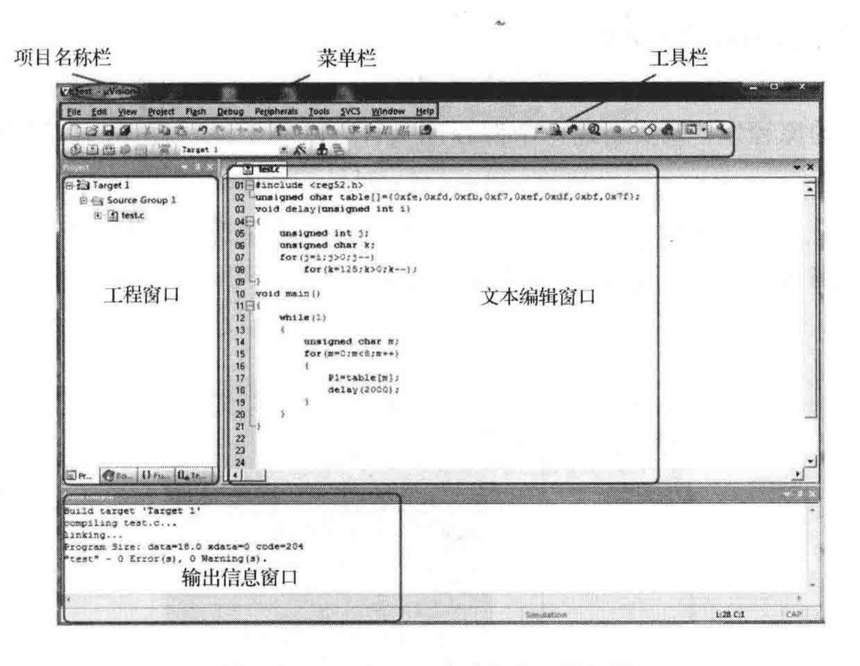


图 9－8　Keil μVision4 集成开发环境

1. 创建 Keil 工程文件

Keil μVision4 是通过工程项目的方法来管理文件的，而不是单一文件的模式。所有文件包括源程序(C 程序或汇编程序)、头文件，甚至说明性技术文档都可以放在工程项目文件里统一管理。

运行 Keil μVision4 后，按照以下步骤建立一个工程项目。

(1) 单击 Project(工程)菜单，在弹出的下拉菜单中选择“New μVision Project”(新工程)命令，如图 9－9 所示。

(2) 在弹出的“Creat New Project”对话框的文件名文本框中输入一个 C 程序(或汇编程序)工程项目的名称，不需要扩展名，将其保存到指定位置。对于已有的工程文件，可以用“Open Project”加载。

(3) 选择所需的单片机器件，这里选择较常用的 Atmel 公司的 AT89C51，此时对话框如图 9－10 所示，AT89C51 的功能、特点显示在右栏中。

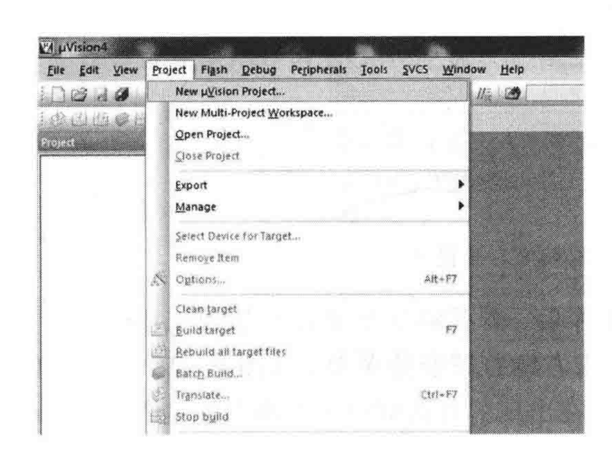


图 9－9　Project 菜单

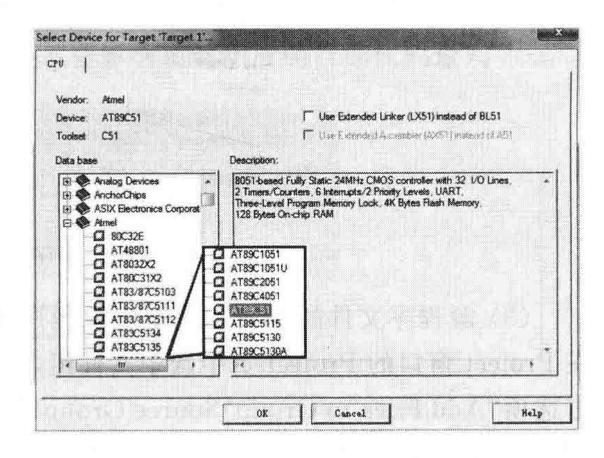


图 9－10　单片机型号选取界面

2. 在工程中添加源程序文件

可以通过菜单命令“File→Open”或快捷键“Ctrl+O”在工程中加入已有的源程序文件。如果没有现成的源程序，就要新建一个源程序文件，以便于编写程序。以下介绍如何新建一个源程序文件。

(1) 源程序文件的创建。单击图 9-11 中的“New”快捷按钮(即新建文件)，在右侧会出现一个新的编辑窗口。这个操作也可以通过菜单命令“File→New”或快捷键“Ctrl+N”来实现。

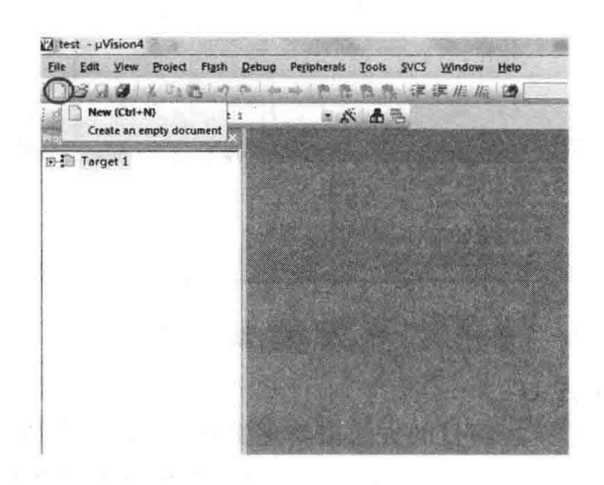


图 9-11　工程中创建新的源程序文件

(2) 源程序文件的保存。新建完编辑窗口后，单击“Save”按钮，或通过菜单命令“File→Save”或快捷键“Ctrl+S”进行保存。在出现的如图 9-12 所示的窗口中命名一个文件名，然后保存。**汇编语言源程序文件命名的后缀为.asm** 。



图 9-12　保存一个汇编语言源程序

(3) 源程序文件的添加。在创建源程序文件后，需要将这个文件添加到工程项目中。在 Project 窗口的 Project 页中选中文件组，单击右键打开快捷菜单，如图 9-13 所示，单击选项“Add Files to Group‘Source Group1’”，在出现的对话框中选中刚刚创建的test.asm (也可以是已有源程序文件)，这时在 Source Group1 文件夹图标左侧出现了“+”号，表明文件组中添加了文件，单击便可以展开查看。

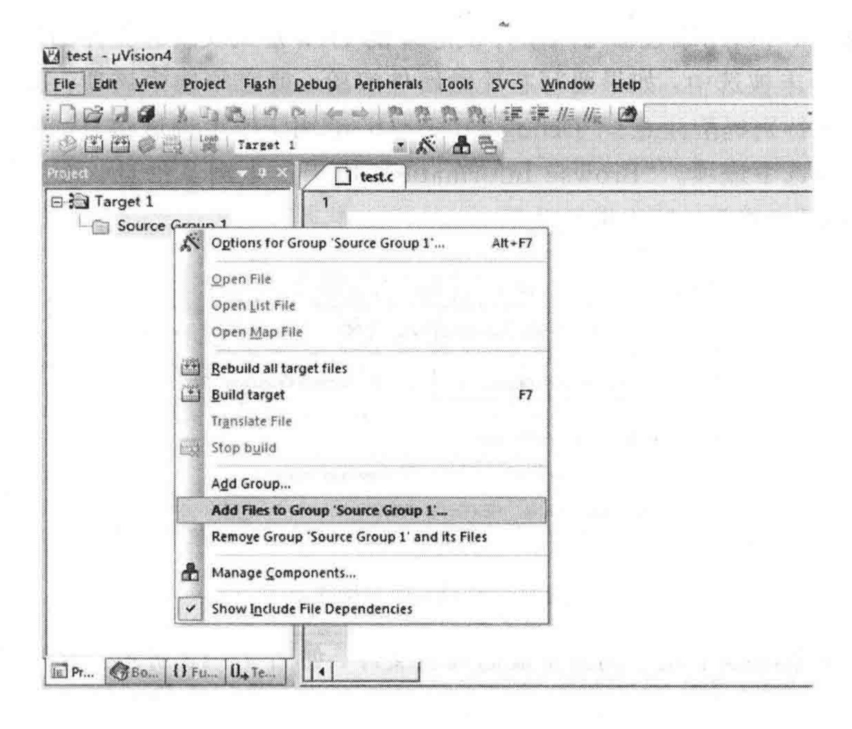


图 9-13 添加文件到源代码组 1 的界面

（4）源程序文件的查看。在源程序编辑窗口查看刚才加入的文件，如图 9-19 所示。

3. 工程的详细设置

工程项目建立好以后，还要对工程进行进一步的设置，以满足要求。

首先单击 Project 窗口的 Target1，然后使用菜单“Project→Options for Target‘Target1’”，即出现对工程设置的对话框，这个对话框共有 11 个页面，在此只介绍其主要功能。

（1）Target 标签页：用于存储器设置，如图 9-14 所示。“Memory Model”用于设置 RAM 空间的使用，主要用于 C51 语言，有三个选择项：Small 是所有变量都在单片机的内部 RAM 中；Compact 可以使用一页外部扩展 RAM；Large 可以使用全部外部扩展 RAM。“Code Rom Size”用于设置 ROM 空间，同样也有三个选择项：Small 模式，只使用低于 2 KB的程序空间；Compact 模式，单个函数的代码量不能超过 2 KB，整个程序可以使用 64 KB 的程序空间；Large 模式，可使用全部 64 KB 的程序空间。“Use On-chip ROM”用于确认是否仅使用片内 ROM(注意：选中该项并不会影响最终生成的目标代码量)。

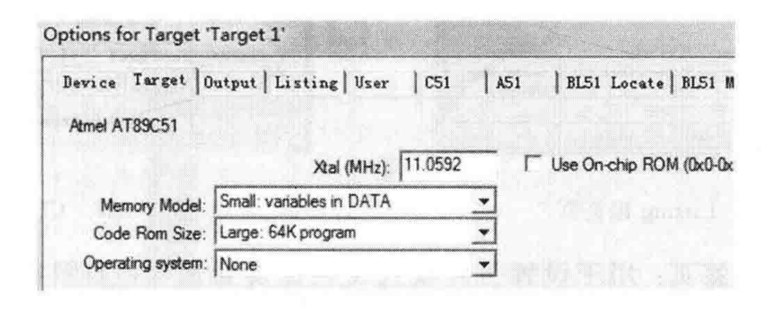


图 9-14 Target 标签页

（2）Output 标签页：用于输出选项设置，如图 9-15 所示。“Creat Hex File”用于生成

可执行代码文件(可以用编程器写入单片机芯片的 Hex 格式文件，文件的扩展名为.hex)，默认情况下该项未被选中，如果要写芯片做硬件实验，就必须选中该项，这一点是初学者易疏忽的，在此特别提醒注意。“Debug Information”用于产生调试信息，如果需要对程序进行调试，应当选中该项。“Browse Information”用于产生浏览信息，该信息可以用菜单“View→Browse”来查看，这里取默认值。

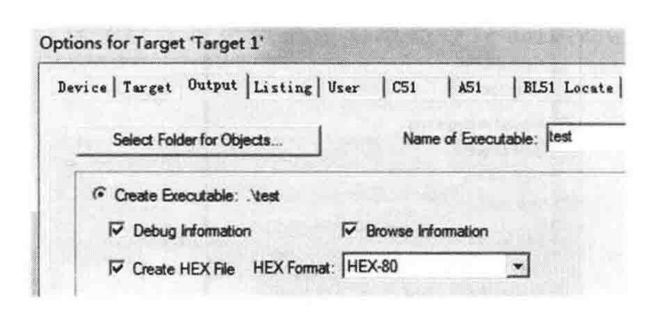


图 9-15　Output 标签页

(3) Listing 标签页：用于调整生成的列表文件，如图 9-16 所示。在汇编或编译完成后将产生(*.lst)的列表文件，在连接完成后也将产生(*.m51)的列表文件，该页用于对列表文件的内容和形式进行细致的调节，其中比较常用的选项是“C Compiler Listing”下的“Assembly Code”项，选中该项可以在列表文件中生成 C 语言源程序所对应的汇编代码。

(4) C51 标签页：用于对 Keil 中 C51 编译器的编译过程进行控制，如图 9-17 所示。其中比较常用的是“Code Optimization”组，该组中 Level 是优化等级，C51 在对源程序进行编译时，可以对代码进行多至 9 级的优化，默认使用第 8 级，一般不必修改，如果在编译中出现一些问题，可以降低优化级别。Emphasis 是选择编译优先方式的，第一项是代码量优先(最终生成的代码其代码量小)，第二项是速度优先(最终生成的代码其代码速度快)，第三项是缺省，默认速度优先，可根据需要更改。

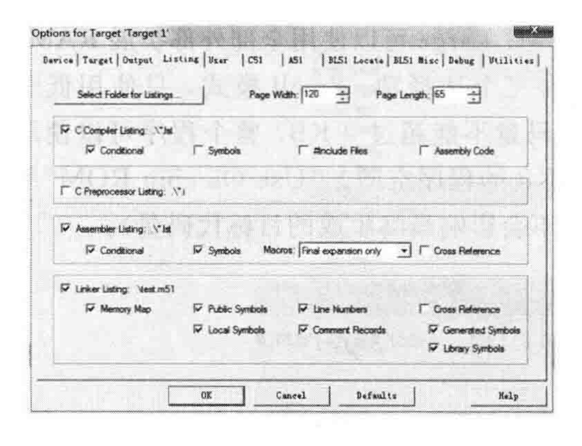


图 9-16　Listing 标签页

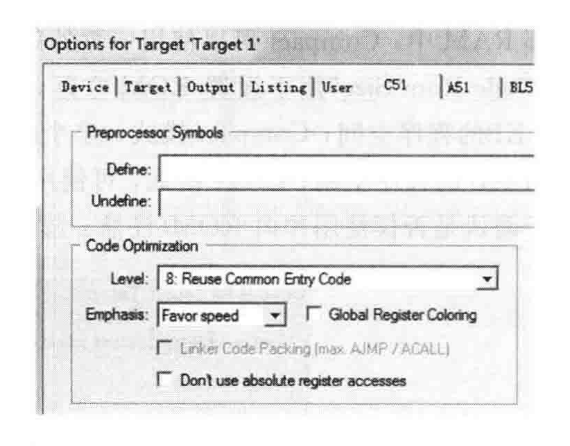


图 9-17　C51 标签页

(5) Debug 标签页：用于设置仿真模式及调试设置选项，如图 9-18 所示。Keil μVision4 的两种仿真模式分别是软件模拟和硬件仿真。软件模拟选项是将 Keil μVision4 调试器设置成软件模拟模式，在此模式下不需要实际的目标硬件就可以模拟 MCS-51 单片机的很多功能(例如第 4 章的程序)，在制作硬件电路之前就可以测试应用程序，非常有

用。硬件仿真则需要和仿真器联合使用(详见 9.2.3 小节)。

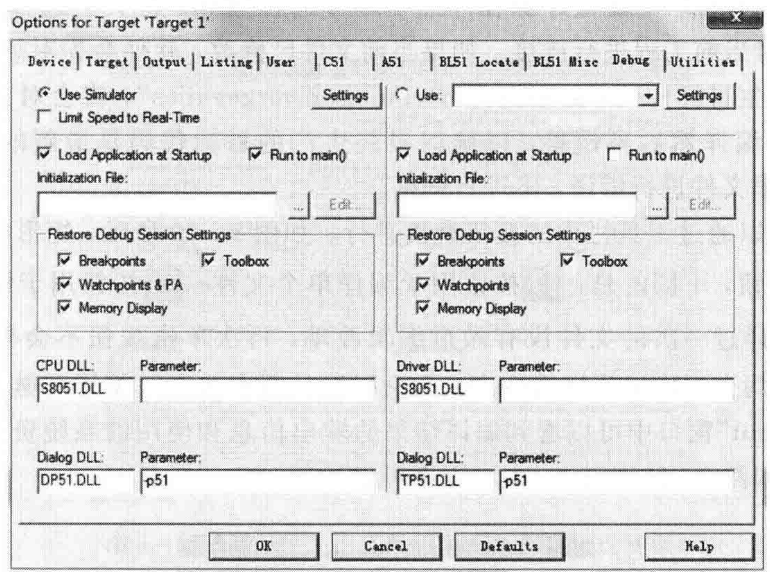


图 9-18　Debug 标签页

9.2.2　应用程序的编辑、编译和链接

1. 源程序的编辑和修改

除了通过添加源程序建立编辑环境外，还可以采用其他方法编写程序。例如，把第 4 章例 4-1 的参考程序通过一个独立编辑器(可以是写字板或记事本)录入，并存为 test.asm (汇编程序扩展名必须是 asm)，通过"Add Files to Group'Source Group1'"选项加载，加载后的界面如图 9-19 所示。如果是一个新程序，我们也可以在编辑窗口中直接编写程序。对于加载完的源程序，可以在编辑窗口直接修改。

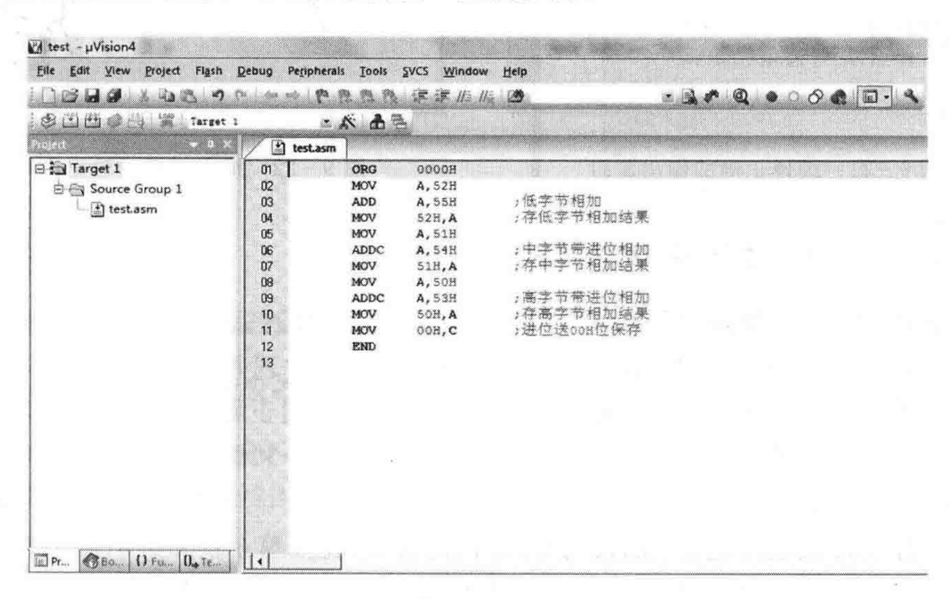


图 9-19　源程序编辑窗口

2. 编译和链接

在设置好工程文件并编写好源程序后，即可进行编译、链接。选择菜单“Project→Build target”，对当前工程进行链接。如果当前文件已修改，软件会先对该文件进行编译，然后再链接以产生目标代码；如果选择“Rebuild all target files”，将会对当前工程中的所有文件重新进行编译然后再链接，以确保最终生产的目标代码是最新的；而“Translate Files”项则仅对该文件进行编译，不进行链接。

以上操作可以通过工具栏中的按钮直接进行。如图 9－20 所示，矩形框中最左边三个图标都是编译按钮，不同的是：按钮用于编译单个文件；按钮用于编译链接当前项目，如果先前编译过一次的文件没有做过编辑改动，再次单击按钮不会再次编译；按钮，每单击一次均会再次编译链接一次，不论程序是否有改动。编译和链接完成后，在下方的“Build Output”窗口中可以看到编译结果的输出信息和使用的系统资源情况等，如果没有语法错误（如图 9－21 所示），表示程序可以运行。

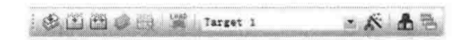


图 9－20　编译按钮

如果源程序中有语法错误，在“Build Output”窗口中会有错误报告出现，如图 9－22 所示（表示源程序第 5 行有语法错误），这时没有生成目标代码，**源程序中存在语法错误，程序不能运行，必须改正错误**。双击错误示意行，可以在编辑器窗口中定位到源程序相应的位置。对源程序反复修改之后，最终会得到如图 9－21 所示的正确结果，提示获得了名为 test.hex 的文件（可以被编程器读入并烧录到 ROM 芯片中），同时还产生了一些其他相关的文件，可用于仿真与调试，这时可以进入下一步调试的工作。

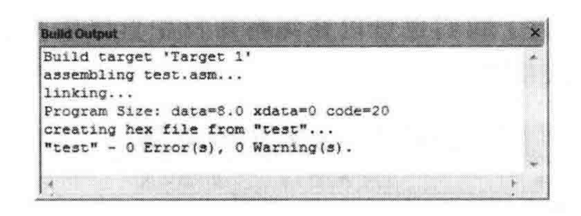


图 9－21　编译正确的输出窗口

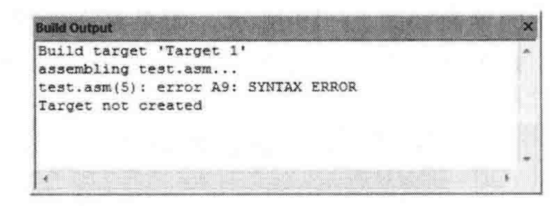


图 9－22　编译错误的输出窗口

9.2.3　应用程序的仿真和调试

通过编译（汇编）和链接，可以获得目标代码，但是做到这一步仅仅代表源程序没有语法错误。**源程序中是否存在逻辑错误，必须通过调试才能发现并解决**。事实上，除了极简单的程序以外，绝大部分的程序都要通过反复调试才能得到正确的结果，因此，调试是软件开发中一个重要环节。下面通过实例介绍常用的调试命令和调试方法。

1. 仿真器的连接

如果程序不涉及硬件电路，而仅与 CPU 内部有关（比如第 3 章和第 4 章的例程），则可以只使用 Keil μVision4 集成开发环境的软件模拟器。如果程序与硬件电路有关，就必须使用硬件仿真器，并且和 Keil μVision4 集成开发环境一起使用。在使用硬件仿真器之前，需要进行仿真环境的设置。

(1) 选择硬件仿真器类型。打开 Debug 标签页设置窗口，选中右侧一栏 Use，并在下拉菜单中选择第一个硬件仿真器 Keil Monitor-51 Driver，如图 9 - 23 所示。

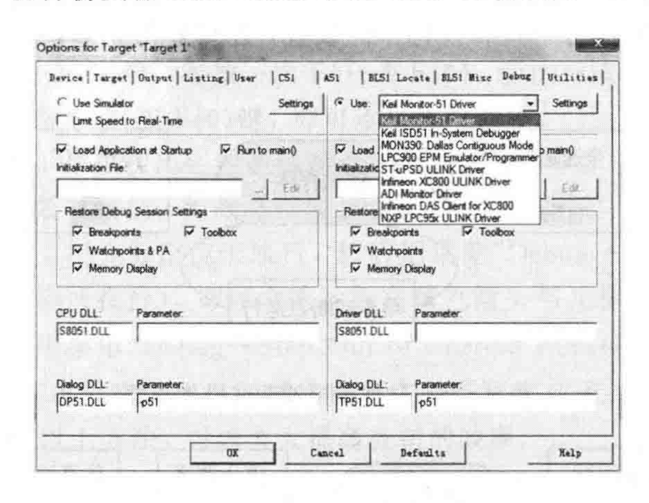


图 9 - 23 使用硬件仿真器的 Debug 标签页设置窗口

(2) 查找串口号。连接仿真器时需注意和计算机串口匹配的问题，因而需要确定计算机的串口号。打开计算机的设备管理器即可查询计算机的串口号，如图 9 - 24 所示。

(3) 设置串口波特率。在确定了计算机所使用的串口号之后，就可以继续对硬件仿真器进行设置。单击 Debug 标签页设置窗口右栏中的 Settings，对计算机所使用的串口号和波特率进行设置，串口号与设备管理器上所显示的一致，波特率可以选择最大值 115 200 b/s，如图 9 - 25 所示。

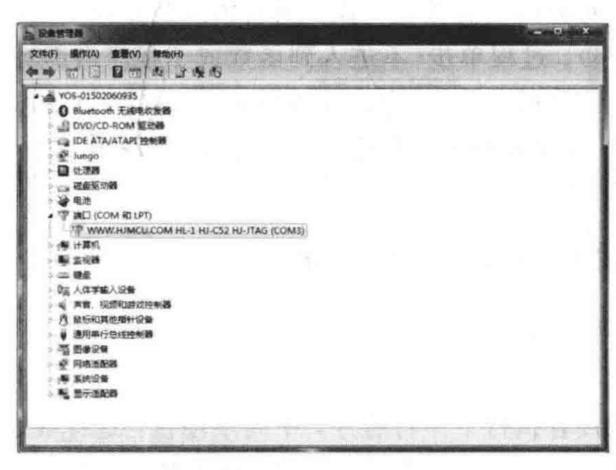
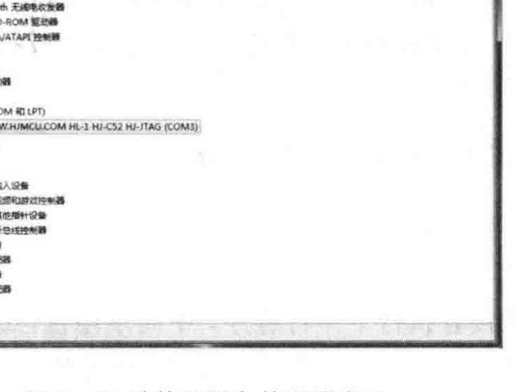



图 9 - 24 计算机设备管理器窗口

图 9 - 25 串口号和波特率设置窗口

(3) 仿真器的连接。拔下单片机应用系统(用户板)的 CPU，改插仿真器的仿真头，将仿真器通过串行接口(或 USB 口)连接到计算机上。

2. 仿真的启动和停止

1) 使用软件模拟器

使用软件模拟器进行仿真时，需要先进入工程设置的 Debug 标签页，将仿真方式设置成软件模拟器仿真，然后进入调试过程，如图 9 - 26 所示。

2）使用硬件仿真器（真实仿真器）

使用硬件仿真器进行仿真时，需要先进入工程设置的 Debug 标签页，将仿真方式设置成硬件仿真器仿真，并进行串口和波特率的设置，连接好仿真器，然后进入调试过程，如图 9－27 所示。

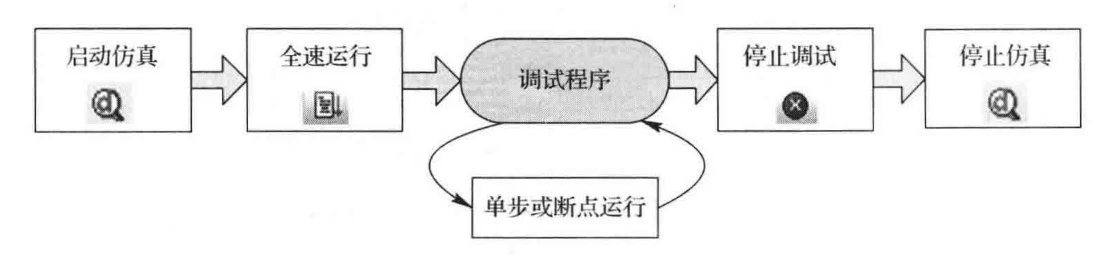


图 9－26　软件模拟器调试操作步骤

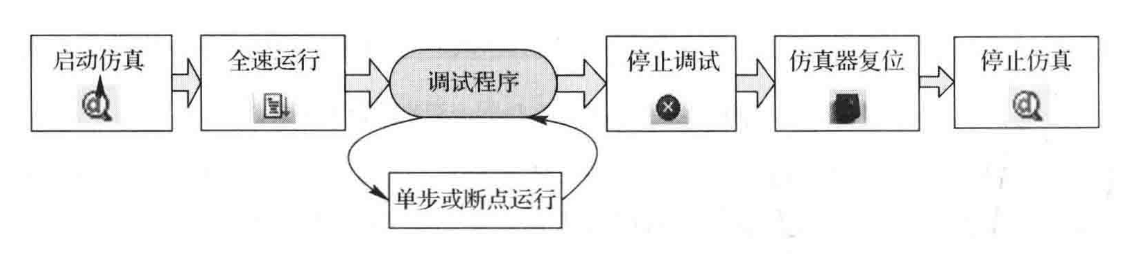


图 9－27　硬件仿真器调试操作步骤

3. 常用调试命令

进入调试状态后，界面与编缉状态相比有明显的变化，Debug 菜单项中原来不能用的命令现在已可以使用了，工具栏会多出一个用于运行和调试的工具条。如图 9－28 所示，Debug 菜单上的大部分命令可以在此找到对应的快捷按钮，从左到右依次是复位、全速运行、暂停、单步（进入到函数或子程序内部）、过程单步（不进入到函数或子程序内部）、跳出函数或子程序（只有软件仿真时有效）、运行到当前行（光标位置处）、显示光标位置、记录运行轨迹、观察运行轨迹、反汇编窗口、变量观察窗口、代码作用范围分析、内存窗口、性能分析（只有软件仿真支持）、工具按钮等命令。

图 9－28　调试工具条

1）全速运行

全速运行是指一行程序执行完以后紧接着执行下一行程序，中间不停止，这样程序执行的速度很快，只能看到该段程序执行的总体效果，即最终结果正确还是错误，但如果程序有错，则难以确认错误出现在哪些程序行。图 9－28 的第 2 个选项即是全速运行。也可以通过 Debug 菜单项中的 Run 实现全速运行。

2）单步运行

单步运行是每次执行一行程序，执行完该行程序以后即停止，等待命令执行下一行程序，此时可以观察该行程序执行完以后得到的结果是否与我们编写该行程序前的预期结果相同，借此可以找到程序中的问题所在。

使用 Debug 菜单项中的 Step 选项或相应的命令按钮或使用快捷键 F11 可以单步运行

程序。使用菜单 Step Over 或功能键 F10 可以通过过程单步形式运行程序。所谓过程单步，是指将汇编语言中的子程序或高级语言中的函数作为一个语句来全速运行。

按下 F11(或 F10)键，可以看到源程序窗口的左边出现了一个黄色调试箭头，指向源程序的第一行，如图 9-29 所示。每按一次 F11(或 F10)，即执行该箭头所指程序行，然后箭头指向下一行，不断按 F11(或 F10)键，即可逐步执行程序。

通过单步运行程序，可以找出一些逻辑错误，但是仅依靠单步运行来查错有时是困难的，或虽能查出错误但效率很低，为此必须辅之以其他的方法：第一，用鼠标在子程序的最后一行(RET)点一下，把光标定位于该行，然后用菜单"Debug→Run to Cursor line"或工具按钮 (执行到光标所在行)，即可全速运行完黄色箭头与光标之间的程序行；第二，在进入该子程序后，使用菜单"Debug→Step Out of Current Function"或工具按钮 (单步执行到该函数外)，即可全速运行完调试光标所在的子程序或子函数并指向主程序中的下一行程序。灵活应用以上方法，可以大大提高查错的效率。

3) 断点运行

程序调试时，已经确定正确的程序段不必每次都单步运行，或者一些程序行必须满足一定的条件才能被执行到(如程序中某变量达到一定的值、按键被按下、串口接收到数据、有中断产生等)，这些条件往往是异步发生或难以预先设定的，这类问题使用单步运行的方法是很难调试的，这时就要使用到程序调试中的另一种非常重要的方法——断点运行。

断点设置的方法有多种，常用的是在某一程序行设置断点。在程序行设置/移除断点的方法是：将光标定位于需要设置断点的程序行，使用菜单"Debug→Insert/Remove BreakPoint"设置或移除断点(也可以用鼠标在该行双击实现同样的功能)，如图 9-30 所示。

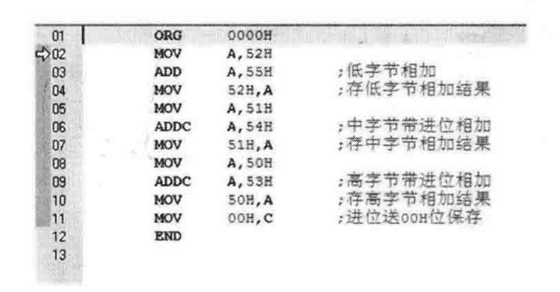


图 9-29 源程序窗口调试状态

```
01        ORG     0000H
02        MOV     A,52H
03        ADD     A,55H       ;低字节相加
04        MOV     52H,A       ;存低字节相加结果
05        MOV     A,51H
06        ADDC    A,54H       ;中字节带进位相加
07        MOV     51H,A       ;存中字节相加结果
08        MOV     A,50H
09        ADDC    A,53H       ;高字节带进位相加
10        MOV     50H,A       ;存高字节相加结果
11        MOV     00H,C       ;进位送00H位保存
12        END
13
```

图 9-30 断点的设置和移除

设置好断点后可以全速运行程序，一旦执行到该程序行即停止，可在此观察有关变量值，以确定问题所在。

9.2.4 应用程序调试的常用窗口

Keil μVision4 在调试程序时提供了多个窗口，主要包括输出窗口(Output Windows)、观察窗口(Watch Windows)、存储器窗口(Memory Windows)和工程窗口寄存器页等。进入调试模式后，可以通过菜单 View 下的相应选项打开或关闭这些窗口。在程序调试过程中，可以充分利用 Keil μVision4 提供的各种窗口，来提高程序调试的效率。

1. 输出窗口

图 9-31 所示为输出窗口。进入调试程序后，输出窗口自动切换到 Command 页，该

页用于输入调试命令和输出调试信息。

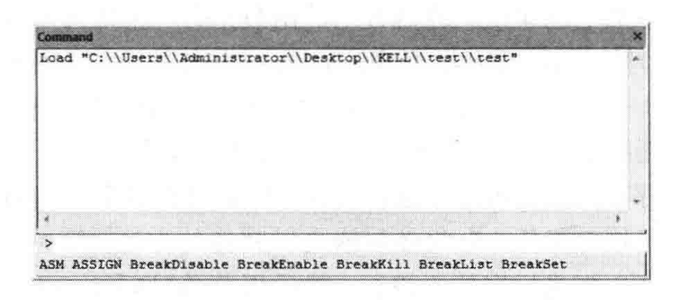


图 9-31　输出窗口

2. 存储器窗口

用图 9-32 所示方法打开存储器窗口，窗口中可以显示单片机中 RAM 和 ROM 的值。通过在 Address 编辑框内输入“字母：数字”即可显示从这个数字开始的若干存储单元的值，字母可选择 C、D、I、X，其中：

- C：代表 ROM 空间；
- D：代表直接寻址的片内 RAM 空间；
- I：代表间接寻址的片内 RAM 空间；
- X：代表扩展的外部 RAM 空间；
- 数字：代表想要查看的地址（习惯上采用十六进制，在数字后面要加上 H）。

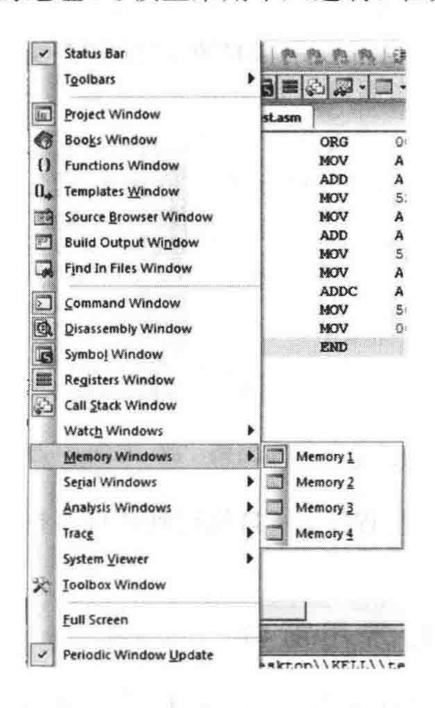


图 9-32　通过菜单项打开存储器窗口

例如：输入“D：30H”即可观察到地址 30H 开始的片内 RAM 单元中的值；输入“C：0000H”即可显示从 0000H 开始的 ROM 单元中的值，即查看程序的二进制代码。RAM 或 ROM 单元中的值可以在窗口中以各种形式显示，如十进制、十六进制、字符型等。改变显示方式的方法是单击鼠标右键，在弹出的快捷菜单中选择，默认的是十六进制显示方式。

3. 工程窗口寄存器页

图9-33所示为工程窗口寄存器页的内容。寄存器页包括了当前的工作寄存器组和系统寄存器组。系统寄存器组有一些是实际存在的寄存器(如A、B、DPTR、SP、PSW等)，有一些是实际中并不存在或虽然存在却不能对其操作的(如PC、States等)。每当程序中执行到对某寄存器的操作时，该寄存器会以反色(蓝底白字)显示，鼠标单击后按下F2键，即可修改该值。存储器窗口和工程窗口寄存器页是汇编程序调试过程中常用的窗口。

4. 观察窗口

图9-34所示为观察窗口。由于工程窗口中仅可以观察到工作寄存器和有限的寄存器(如A、B、DPTR等)，如果需要观察其他的寄存器的值或在高级语言编程时需要直接观察变量，就要借助于观察窗口。一般情况下，仅在单步运行时才对变量的值的变化感兴趣。全速运行时，变量的值是不变的，只有在程序停下来之后，才会将这些值的最新变化反映出来。但是，在一些特殊场合下也可能需要在全速运行时观察变量的变化，此时可以单击"View→Periodic Window Update"(周期更新窗口)，确认该项处于被选中状态后，即可在全速运行时动态观察有关值的变化。但是，选中该项，将会使程序模拟执行的速度变慢。观察窗口是C程序调试过程中常用的窗口之一。

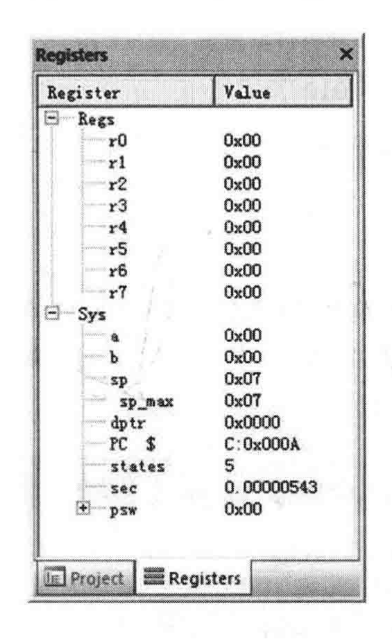


图9-33 工程窗口寄存器页

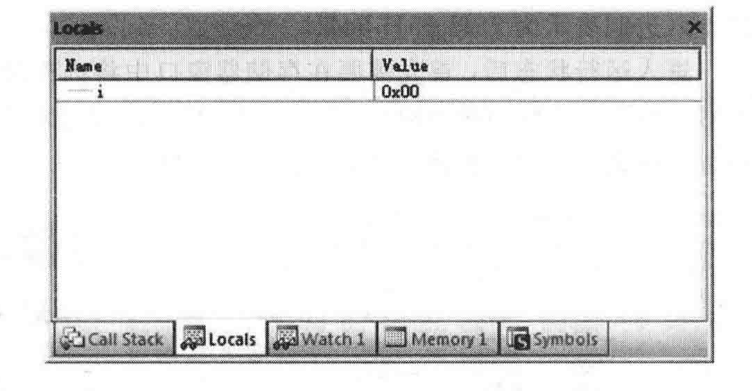


图9-34 观察窗口

9.2.5 调试实例

下面以第4章的例4-1为例，全面讲述程序调试方法。

三字节无符号数相加，其中被加数在内部RAM的50H、51H和52H单元中(低位在后)，加数在内部RAM的53H、54H和55H单元中(低位在后)，要求把相加之和存放在50H、51H和52H单元中(低位在后)，进位存放在位寻址区的00H位中。

参考程序：

```
    ORG     0000H
    MOV     A, 52H
    ADD     A, 55H      ; 低字节相加
    MOV     52H, A      ; 存低字节相加结果
    MOV     A, 51H
    ADDC    A, 54H      ; 中字节带进位相加
    MOV     51H, A      ; 存中字节相加结果
    MOV     A, 50H
    ADDC    A, 53H      ; 高字节带进位相加
    MOV     50H, A      ; 存高字节相加结果
    MOV     00H, C      ; 进位送 00H 位保存
    END
```

(1) 编译程序。如上述参考程序所示，为了演示“程序的调试”，我们首先给源程序制造一个错误，将第 6 行中的带进位的加指令的 ADDC 改为 ADD，然后进行编译，如图 9-35所示。由于程序中并无语法错误，所以编译时不会有任何出错提示。但由于中间字节相加结果没有考虑进位位，所以可能会导致运算结果出错。

(2) 演算或预判给出正确结果。我们设定 24 位的被加数和加数均为 10101010 10101010 10101010，则相加的结果如下：

```
      10101010   10101010   10101010
+     10101010   10101010   10101010
----------------------------------------
    1 01010101   01010101   01010100
```

用十六进制表示为 55H 55H 54H。

进入运行状态后，首先需要在存储器窗口中将内部地址单元 50H 到 55H 的值修改为 AAH(即输入加数和被加数)，其方法是：用鼠标双击对应存储器窗口中的存储单元，单元中数据被选中后便可修改数据，如图 9-36 所示。

```
test.asm
01        ORG     0000H
02        MOV     A,52H
03        ADD     A,55H      ;低字节相加
04        MOV     52H,A      ;存低字节相加结果
05        MOV     A,51H
06        ADD     A,54H      ;中字节带进位相加
07        MOV     51H,A      ;存中字节相加结果
08        MOV     A,50H
09        ADDC    A,53H      ;高字节带进位相加
10        MOV     50H,A      ;存高字节相加结果
11        MOV     00H,C      ;进位送00H位保存
12        END
```

图 9-35　有逻辑错误的调试窗口

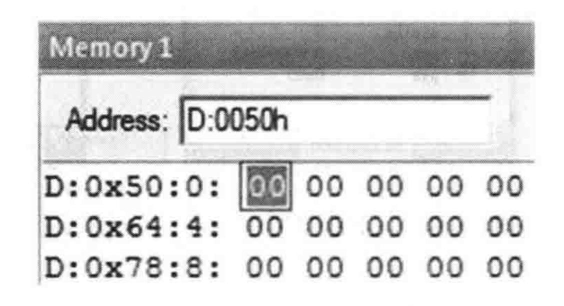


图 9-36　存储器值修改

(3) 运行程序并查看程序运行结果。先全速运行，在存储器窗口中输入“D：0050h”，查看结果为 55H 54H 54H，如图 9-37 所示。这与我们上面计算的结果不同，说明出现了错误。为此需要单步运行，通过中间结果找到错误。

需要注意的是：有些情况下字节之间相加没有进位，比如：两个 01010101 01010101 01010101 相加，上面错误则无法发现，这就需要我们在调试时考虑到所有的情况。

（4）**单步运行调试，观察中间值。**进入单步运行状态后，按 F10 以过程单步执行程序，观察寄存器窗口的各寄存器的运算结果。按照预置的数据其输出结果在存储器地址单元 50H 到 52H 的结果应该为“55H、55H、54H”，而实际调试中当程序执行完第 7 行时，发现存储器地址 51H 中的结果为“54H”，与预期结果不符(如图 9－38 所示)，并且观察到单元 52H 和 55H 加法完成后寄存器窗口中状态字 PSW 的进位位 Cy 为 1(如图 9－39 所示)，而在单元 51H 和 54H 相加时却没有加上此进位位，最终找到逻辑错误。

（5）**改正错误并验证结果。**检查源程序发现第 6 行中加法为 ADD，不含进位加法指令，当改成 ADDC 后发现结果与预期一致，如图 9－40 所示。

```
Memory 1
Address: D:0050h
D:0x50:0:  55 54 54 AA AA AA 00 00 00 00 00 00
D:0x65:5:  00 00 00 00 00 00 00 00 00 00 00 00
D:0x7A:A:  00 00 00 00 00 00 FF 07 00 00 00 00
D:0x8F:F:  00 FF 00 00 00 00 00 00 00 00 00 00
D:0xA4:4:  00 00 00 00 00 00 00 00 00 00 00 00
D:0xB9:9:  00 00 00 00 00 00 00 00 00 00 00 00
D:0xCE:E:  00 00 C4 00 00 00 00 00 00 00 00 00
Call Stack | Locals | Watch 1 | Memory 1 | Symbols
```

图 9－37　全速运行得到的错误结果

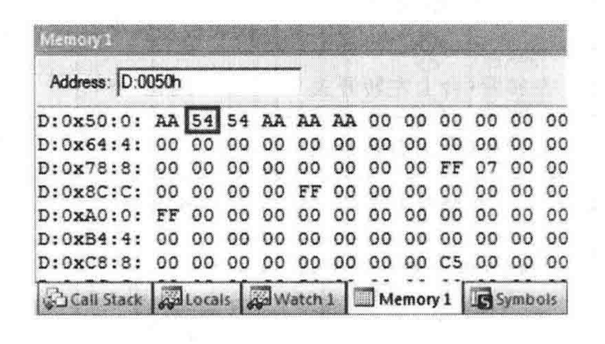


图 9－38　调试过程的中间结果

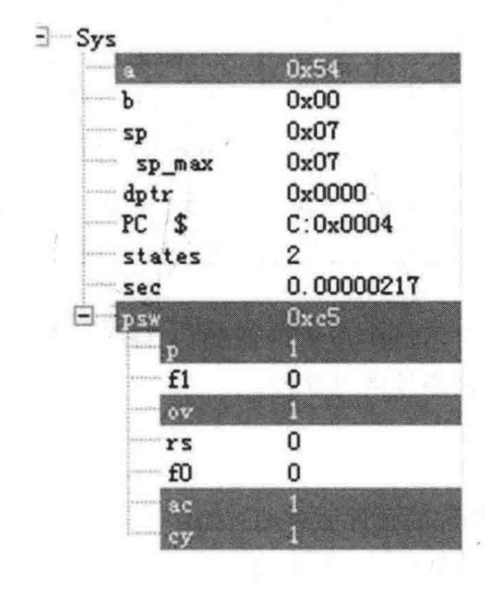


图 9－39　寄存器中状态字显示

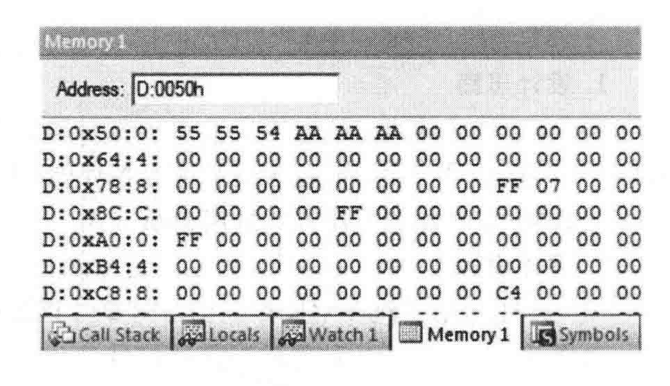


图 9－40　正确显示结果

对于复杂的程序，以上调试过程也许需要重复许多次。

9.3　实际应用案例

9.3.1　汽车驾驶操纵信号灯控制系统

在驾驶汽车时，有左转弯、右转弯、刹车、紧急开关、停靠等操作。在左转弯或右转弯

时，通过转弯操作杆应使左转开关或右转开关合上，从而使左头灯、仪表板左转弯灯、左尾灯或右头灯、仪表板右转弯灯、右尾灯闪烁；合紧急开关时要求前面述及的 6 个信号灯全都闪烁；刹车时，2 个尾灯点亮；若正当转弯时刹车，则转弯时原应闪烁的信号灯仍应闪烁。以上闪烁都是频率为 1 Hz 的低频闪烁，在汽车停靠而停靠开关合上时，左头灯、右头灯、左尾灯、右尾灯按频率为 30 Hz 的高频闪烁。

信号灯应输出的信号如表 9 - 1 所示。

表 9 - 1　汽车驾驶操纵与信号

驾驶操作	输出信号					
	仪表板左转弯灯	仪表板右转弯灯	左头灯	右头灯	左尾灯	右尾灯
左转弯(合上左转开关)	闪烁	—	闪烁	—	闪烁	—
右转弯(合上右转开关)	—	闪烁	—	闪烁	—	闪烁
合紧急开关	闪烁	闪烁	闪烁	闪烁	闪烁	闪烁
刹车(合上刹车开关)	—	—	—	—	亮	亮
左转弯时刹车	闪烁	—	闪烁	—	闪烁	亮
右转弯时刹车	—	闪烁	—	闪烁	亮	闪烁
刹车，并合紧急开关	闪烁	闪烁	闪烁	闪烁	亮	亮
左转弯时刹车，并合紧急开关	闪烁	闪烁	闪烁	闪烁	闪烁	亮
右转弯时刹车，并合紧急开关	闪烁	闪烁	闪烁	闪烁	亮	闪烁
停靠(合停靠开关)	—	—	30 Hz 闪烁	30 Hz 闪烁	30 Hz 闪烁	30 Hz 闪烁

1. 设计思路

将 P1 口作为输出口，利用单片机内部计数器/定时器产生所需的低频(1 Hz)与高频(30 Hz)闪烁信号。设计时应考虑故障监控性能，以提高系统的可靠性。

计数器/定时器的工作方式采用中断方式。

2. 硬件设计

图 9 - 41 所示是汽车驾驶操纵信号灯单片机控制系统的硬件原理图。

由图 9 - 41 可见，各种驾驶操作的信号自 P0 口送入单片机，而使信号灯点亮的输出信号则自 P1 口输出。图中的晶体管是输出驱动级，图的下部是故障监控电路。在 P1.0～P1.5共 6 路输出中，如轮流使其中 1 路的晶体管断开(P1 口相应引脚输出低电平)，这 1 路的信号灯将熄灭，而其他 5 路的晶体管接通(P1 口引脚送来高电平)，相应的信号灯点亮，则在正常情况下，信号灯熄灭的那路将使 P1.7 呈现低电平；如果 P1.7 出现高电平，则说明当前这 1 路出现故障。另外，如使 6 路的晶体管全部接通(P1 口引脚送来高电平)，在正常情况下，P1.7 应呈高电平；如果 P1.7 出现低电平，也说明信号线路存在故障。有故障时，通过软件应使 P1.6 输出高电平，以点亮故障信号灯报警。

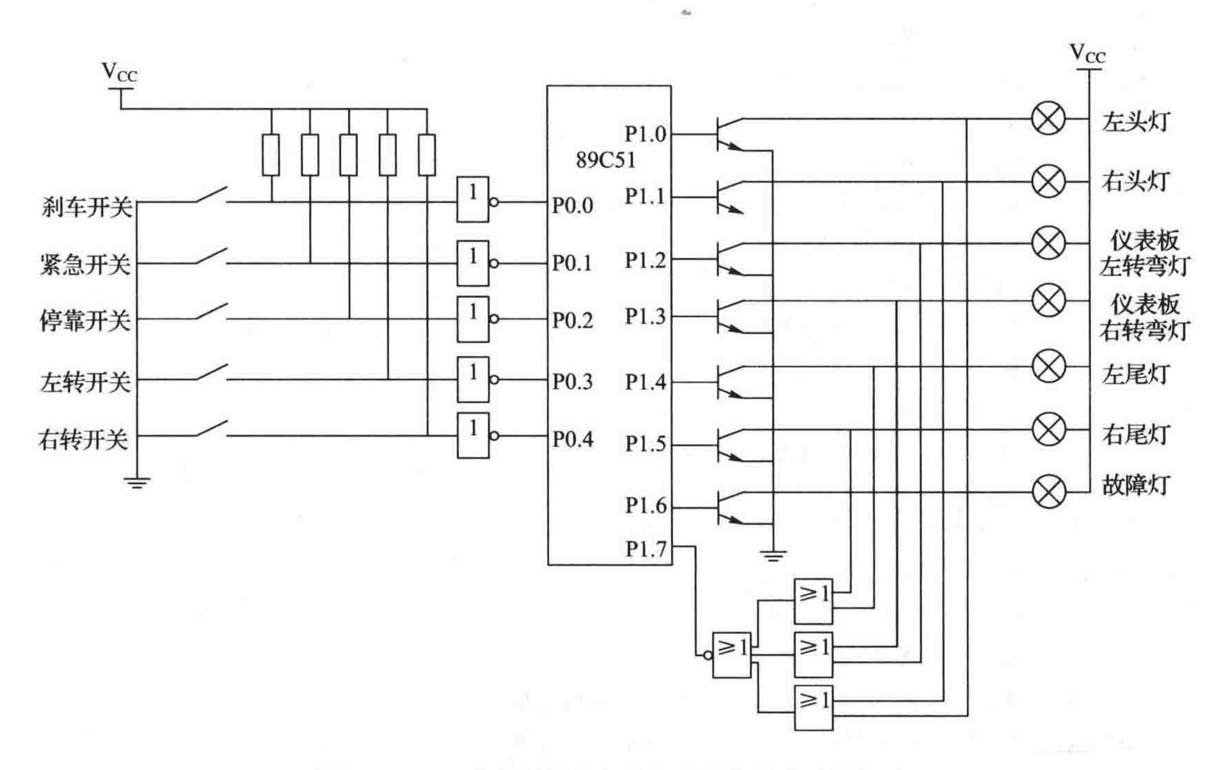


图 9-41　汽车驾驶操纵信号灯单片机控制系统原理

3. 软件设计

参考程序：

```
        ORG     0000H
        LJMP    MAIN                ；跳到主程序
        ORG     000BH               ；定时器/计数器 0 中断入口
        MOV     TH0，#0F0H
        PUSH    PSW                 ；保护现场
        LJMP    INTSUB              ；跳到中断服务程序
MAIN：  MOV     TL0，#0             ；定时器/计数器 0 装载初值
        MOV     TH0，#0F0H
        MOV     TMOD，#00000001B    ；定时器/计数器 0 工作在方式 1
        MOV     20H，#244           ；片内 RAM 20H 单元作计数器，初值设定为 244
        SETB    ET0                 ；定时器/计数器 0 开中断
        SETB    EA                  ；开总中断
        SETB    TR0                 ；启动定时器/计数器 0
        LJMP    $                   ；等待
INTSUB：DJNZ    20H，LAMP           ；片内 RAM 20H 单元未减到 0，转信号灯指示程序段
        MOV     20H，#244           ；片内 RAM 20H 单元已减到 0，则该单元重装载
        MOV     P1，#3FH            ；使 P1.0～P1.5 输出高电平，此为故障监控程序段
        CLR     P1.0
        JB      P1.7，FAULT
        SETB    P1.0
```

```
        CLR     P1.2
        JB      P1.7, FAULT
        SETB    P1.2
        CLR     P1.4
        JB      P1.7, FAULT
        SETB    P1.4
        CLR     P1.1
        JB      P1.7, FAULT
        SETB    P1.1
        CLR     P1.3
        JB      P1.7, FAULT
        SETB    P1.3
        CLR     P1.5
        JB      P1.7, FAULT
        SETB    P1.5
        JB      P1.7, LAMP
FAULT:  SETB    P1.6            ；点亮故障信号灯
LAMP:   MOV     C, 01H          ；信号灯指示程序
        ANL     C, 00H
        ORL     C, 02H          ；以上 3 条使 30 Hz 闪烁信号的占空比为 50%
        ANL     C, P0.2
        MOV     PSW.1, C        ；30 Hz 闪烁信号暂存于 PSW.1
        MOV     C, P0.3
        ORL     C, P0.1
        ANL     C, 07H
        MOV     P1.2, C         ；仪表板左转弯灯闪烁
        MOV     F0, C
        ORL     C, PSW.1
        MOV     P1.0, C         ；左头灯闪烁
        MOV     C, P0.0
        ANL     C, P0.3
        ORL     C, F0
        ORL     C, PSW.1
        MOV     P1.4, C         ；左尾灯亮或闪烁
        MOV     C, P0.4
        ORL     C, P0.1
        ANL     C, 07H
        MOV     P1.3, C         ；仪表板右转弯灯闪烁
        MOV     F0, C
        ORL     C, PSW.1
        MOV     P1.1, C         ；右头灯闪烁
        MOV     C, P0.0
```

```
        ANL     C, P0.4
        ORL     C, F0
        ORL     C, PSW.1
        MOV     P1.5, C         ; 右尾灯亮或闪烁
        POP     PSW             ; 恢复现场
        RETI
        END
```

4. 程序说明

本程序的主程序部分只有 8 条指令，其中 7 条用于初始化，分别为：对 T0 置初值、设定 T0 的工作方式、设定片内 RAM 20H 单元的初值、开 T0 中断、启动 T0、中断等待。

(1) T0 的中断服务子程序：为 T0 重装初值和保护现场后又转去 INTUSB。自 INTUB 起，包含 2 个主要程序段：信号灯指示程序段和故障监控程序段。如 1 s 时间未到，则只执行信号灯指示程序段，根据驾驶操作动作，如转弯、停靠等情形，信号灯将闪烁或点亮(见表 9－1)。每逢 1s 时间到，则先执行故障监控程序段，检查一遍信号指示电路(见图 9－41)，看看是否有硬件故障，然后再执行信号灯指示程序段。

(2) 1 Hz 闪烁信号的产生与占空比：本例令 T0 工作于方式 1，且将初始值预置为 F000H，在 12 MHz 晶振的情形下，每隔 4096 μs 将溢出一次，另以片内 RAM 20H 单元为程序计数器(RAM 20H 单元可以位寻址)，初值置为 244，每逢 T0 溢出一次便减 1；当减到 0 时，经历的时间为 $244 \times 4096\ \mu s \approx 1$ s。在上述 1 s 时间内，片内 RAM 20H 单元最高位不为 1 的时间是 127/244 s，为 1 s 的时间是 $(244-127)/244\ s = 117/244\ s$，故自该位可得占空比接近 50%(实际约为 48%)的 1 Hz 闪烁信号。

9.3.2 高精度模拟信号的采集和显示

实现单路模拟信号采集并显示，要求采用数码管显示，显示位数为 4 位，最小分辨率为 1 mV，模拟输入最大值为 5 V。

1. 任务分析

高精度模拟信号采集和数值显示器由 A/D 转换、数据处理及显示控制等组成。A/D 转换采用集成电路 AD574A。AD574A 是具有单路模拟输入端口的 12 位 A/D 转换器；显示电路采用八段数码管、P1 口动态控制方式。

2. 硬件电路设计

AT89C51 单片机与 AD574A 的接口电路如图 9－42 所示，AD574A 的数据输出通过 P0 数据总线连至 AT89C51，即 P0 口接到 AD574 的数据口，P2 口用于控制工作过程。由于 MCS－51 系列单片机只有 8 位数据口，12 位数据需分两次读进 AT89C51，所以高 8 位和低 4 位需要通过 A0 分别控制输出，这时，$12/\overline{8}$ 接地。当 AT89C51 的 P3.5 查询到 CTS 端转换结束信号后，先将转换后的 12 位 A/D 数据的高 8 位读进 AT89C51，然后再将低 4 位读进 AT89C51。这里不管 AD574A 是否处在启动、转换和输出结果状态，使能端 CE 都必须为 1，因此将 AT89C51 的写控制线 $\overline{WR}$ 和读控制线 $\overline{RD}$ 通过与非门 74LS00 与 AD574A

的使能端 CE 相连。

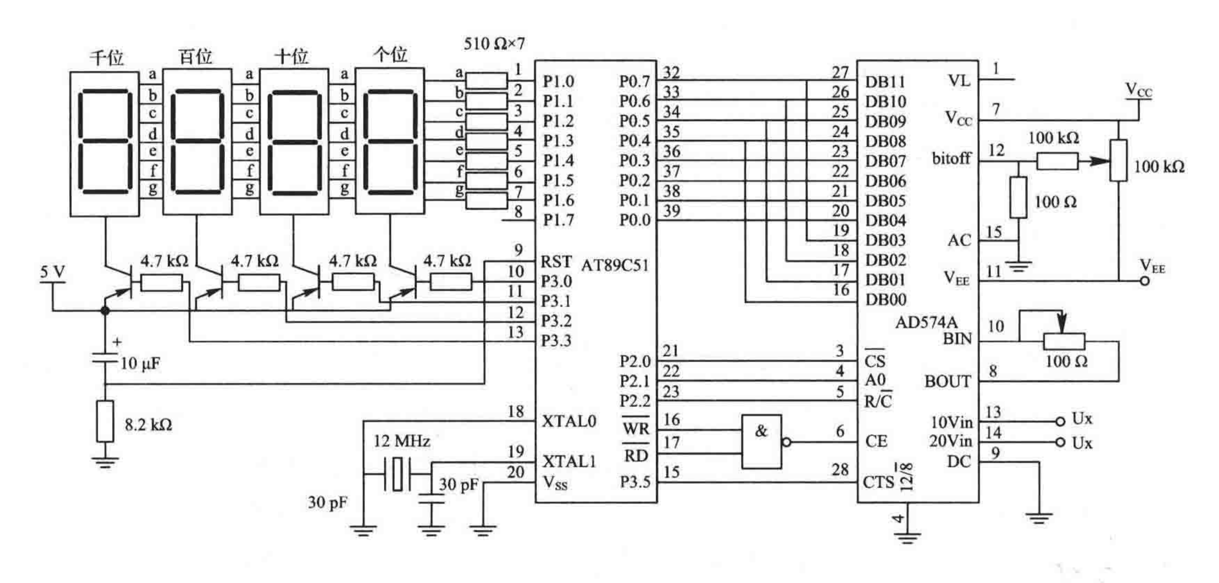


图 9-42　高精度模拟信号采集和数值显示器原理图

3. 软件设计

AD574A 是美国模拟器件公司(Analog)推出的单片高速 12 位逐次比较型 A/D 转换器，内置双极性电路构成的混合集成转换芯片，具有外接元件少、功耗低、精度高等特点，并且具有自动校零和自动极性转换功能，只需外接少量的阻容件即可构成一个完整的 A/D 转换器。AD574A 的主要功能特性如下：

分辨率：12 位；

非线性误差：小于±1/2 LBS 或±1 LBS；

转换速率：25 μs；

模拟电压输入范围：0～10 V 和 0～20 V，0～±5 V 和 0～±10 V 两档四种；

电源电压：±15 V 和 5 V ；

数据输出格式：12 位/8 位；

最大转换数值：4095。

AD574A 的 CE、$12/\overline{8}$、$\overline{CS}$、$R/\overline{C}$ 和 A0 对其工作状态的控制过程如下：

当 CE=1、$\overline{CS}$=0 时，AD574A 才会正常工作。在 AD574 处于工作状态时，当 $R/\overline{C}$=0时进行 A/D 转换；当 $R/\overline{C}$=1 时，进行数据读出。$12/\overline{8}$ 和 A0 端用来控制启动转换的方式和数据输出格式。当 A0=0 时，启动是按完整 12 位数据方式进行的；当 A0=1 时，按 8 位 A/D 转换方式进行。当 $R/\overline{C}$=1，也即当 AD574A 处于数据状态时，A0 和 $12/\overline{8}$ 控制数据输出状态的格式。当 $12/\overline{8}$=1 时，数据以 12 位并行输出；当 $12/\overline{8}$=0 时，数据以 8 位分两次输出。而当 A0=0 时，输出 A/D 转换数据的高 8 位，当 A0=1 时输出 A/D 转换数据的低 4 位，这四位占一个字节的高半字节，低半字节补零。其控制逻辑真值表见表 9-2。

在刚上电时，因 70H～77H 内存单元的数据为 0，则每一通道的数码管显示值都为 0000。当进行一次测量后，将显示出 A/D 转换值。

采用动态扫描法实现 4 位数码管的数值显示。测量所得的 A/D 转换数据放在 70H～71H 内存单元中。测量数据在显示时需转换成为十进制 BCD 码放在 78H～7BH 中。

70H～71H 存放采样值，78H～7BH 存放显示数据，依次为个位、十位、百位、千位。

表 9－2 AD574A 控制端标志意义

CE	$\overline{CS}$	$R/\overline{C}$	$12/\overline{8}$	A0	工作状态
0	×	×	×	×	禁止
×	1	×	×	×	禁止
1	0	0	×	0	启动 12 位转换
1	0	0	×	1	启动 8 位转换
1	0	1	接＋5 V	×	12 位并行输出有效
1	0	1	接 0 V	0	高 8 位并行输出有效
1	0	1	接 0 V	1	低 4 位并行输出有效

A/D 转换器的流程图如图 9－43 和图 9－44 所示。

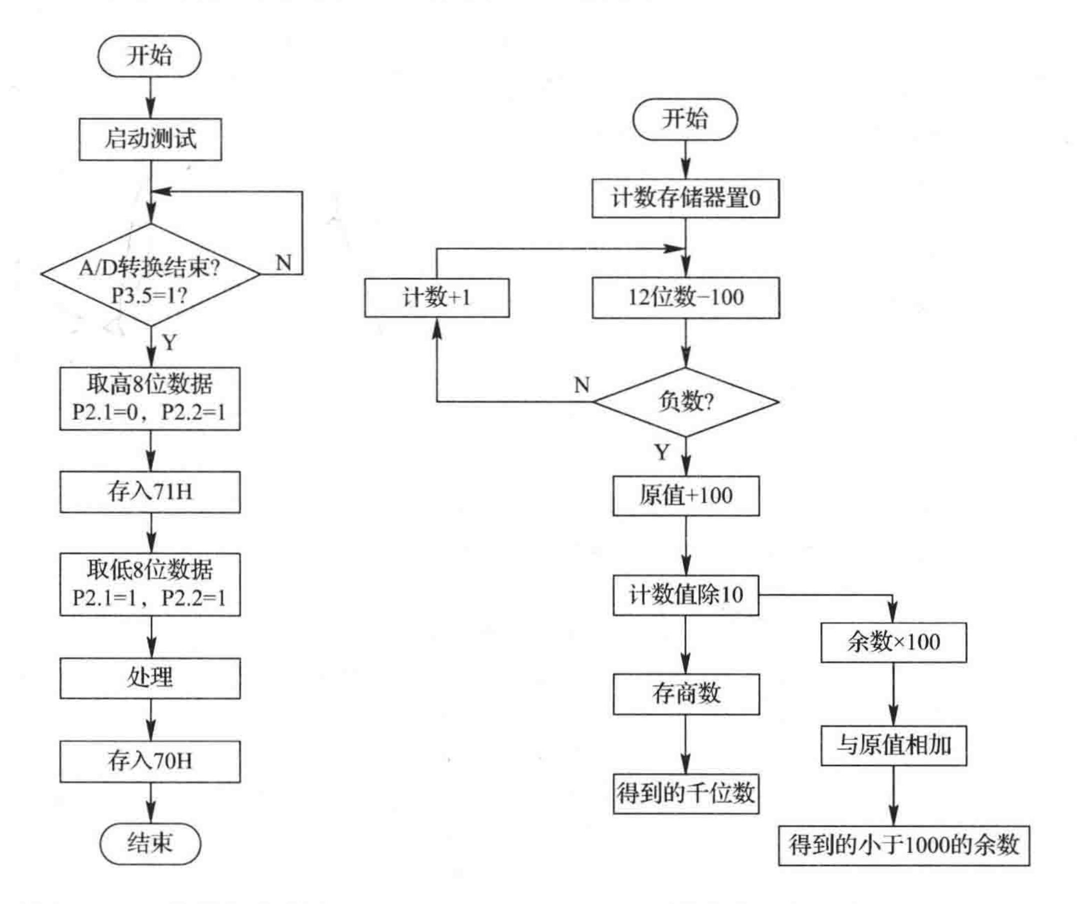


图 9－43 A/D 转换流程图　　9－44 显示数据处理流程图

参考程序：

```
        ORG     0000H       ；程序执行开始地址
        LJMP    START       ；跳至 START 执行
；主程序
START： LCALL   CLEAR       ；初始化
MAIN：  LCALL   DISPLAY     ；显示数据一次
        LCALL   TEST        ；测量一次
        LJMP    MAIN        ；返回 MAIN 循环
        NOP                 ；软件陷阱
        NOP
        NOP
        LJMP    START       ；重新复位启动
CLEAR： CLR     A           ；初始化程序中的各变量
        MOV     P2，A       ；P2 口置 0
        MOV     A，#0FFH
        MOV     P0，A       ；P0、Pl、P3 口置 1
        MOV     P1，A
        MOV     P3，A
        RET
；显示数据处理子程序
DISPLAY：NOP                ；把 A/D 转换数据处理成 12 位，放在 R5R4 中(高 4 位低 8 位)
        MOV     A，71H      ；A/D 转换的高 8 位放入 A 中
        MOV     B，#16      ；显示数据乘以 16
        MUL     AB
        ADD     A，70H
        MOV     R4，A
        MOV     A，#0
        ADDC    A，B
        MOV     R5，A
        NOP                 ；千位数处理
        LCALL   X100        ；查找 100 的倍数
        MOV     A，R6
        MOV     B，#10
        DIV     AB          ；显示数据再除以 10
        MOV     7BH，A      ；千位数放入 7BH 中
        MOV     A，#100
        MUL     AB          ；余数再乘以 100
        ADD     A，R4       ；与原来的低 8 位余数合并
        MOV     R4，A
        MOV     A，#0
        ADDC    A，B
        MOV     R5，A
```

```
         NOP                       ；对百位数进行处理
         LCALL    X100             ；查找 100 的倍数
         MOV      7AH，R6          ；百位数放入 7AH 中
         MOV      A，R4            ；余数放入 A 中
         NOP                       ；对十位数进行处理
         MOV      B，＃10
         DIV      AB
         MOV      79H，A           ；十位数放入 79H 中
         MOV      78H，B           ；个位数放入 78H 中
         MOV      R2，＃0FFH       ；每路显示时间控制 4 ms×255
DIS10：  LCALL    DISP             ；调 4 位 LED 显示程序
         DJNZ     R2，DIS10        ；显示时间控制
         RET
；寻找 100 的倍数的子程序
X100：   NOP      ；寻找 R5R4(16 位)100 的倍数，存入 R6，小于 100 的数存入 R4
         MOV      R6，＃0
X110：   CLR      C
         MOV      A，R4
         SUBB     A，＃100
         MOV      R4，A
         MOV      R5，A
         SUBB     A，＃0
         MOV      R5，A
         CJNE     A，＃0FFH，X120  ；等于＃0FFH，相当于高 8 位多减了 100
         MOV      A，R4
         ADD      A，＃100         ；高 8 位减完了
         MOV      R4，A
         LJMP     X130
X120：   INC      R6
         LJMP     X110
X130：   RET
；LED 共阳显示子程序，显示内容在 78H～7BH 单元中，数据在 P1 口输出，列扫描在
；P3.0～P3.3 口
DISP：   MOV      R1，＃78H        ；赋显示数据单元首址
         MOV      R5，＃0FEH       ；扫描字
PLAY：   MOV      P1，＃0FFH       ；关显示
         MOV      A，R5            ；取扫描字
         ANL      P3，A            ；开显示
         MOV      A，@ R1          ；取显示数据
         MOV      DPTR，＃TAB      ；取段码表首址
         MOVC     A，@A+DPTR       ；查显示数据对应段码
         MOV      P1，A            ；段码放入 P1 口
```

```
        LCALL   DELAY               ；显示 1 ms
        INC     R1                  ；指向下一地址
        MOV     A，P3               ；取 P3 口扫描字
        JNB     ACC.3，ENDOUT       ；4 位显示完转 ENDOUT 结束
        RL      A                   ；扫描字循环左移
        MOV     R5，A               ；扫描字放入 R5 暂存
        MOV     P3，#0FFH           ；显示暂停
        LJMP    PLAY                ；转 PLAY 循环
ENDOUT：MOV     P3，#0FFH           ；显示结束，端口置 1
        MOV     P1，#0FFH
        RET                         ；子程序返回
；LED 数码显示管用共阳段码表，分别对应 0～9，最后一个是"熄灭符"
TAB：   DB      0C0H，0F9H，0A4H，0B0H，99H，92H
        DB      82H，0F8H，80H，90H，0FFH
；1 ms 延时子程序，LED 显示用
DELAY： MOV     R6，#14H
DL1：   MOV     R7，#19H
DL2：   DJNZ    R7，DL2
        DJNZ    R6，DL1
        RET
；模/数转换测量子程序
TEST：  MOV     R0，#70H
        CLR     P3.5
        MOV     A，#00000000B       ；置 AD574 转换控制位
        MOV     P2，A               ；启动 12 位转换
WAIT：  JB      P3.5，TEST1         ；等 A/D 转换结束信号发出后转 TEST1
        LJMP    WAIT                ；P3.7 为 0，等待
TEST1： MOV     A，#00000100B       ；置高 8 位输出有效
        MOV     P2，A
        MOV     A，P0               ；高 8 位读入 A 中
        MOV     71H，A
        MOV     A，#00000110B       ；置低 4 位输出有效
        MOV     A，P0               ；低 4 位读入 A 中
        RL      A
        RL      A
        RL      A
        RL      A
        MOV     70H，A              ；转化数据低 4 位放入 70H 的低 4 位
        RET
        END
```

思考与练习

1. 单片机系统设计分哪几部分？
2. 单片机系统调试分为几个步骤？
3. 单片机开发工具主要有什么？简述各工具的具体用途。
4. 为什么需要对源程序进行汇编或编译？单片机能够直接运行的是什么代码？
5. 如何用 Keil μVision4 创建一个应用程序？
6. 为什么要使用断点调试方法？如何设置断点？
7. 应用程序调试的常用窗口有哪几个？如何使用？
8. 试设计一个电子钟，实现以下功能：

(1) 可以显示年、月、日、时、分、秒等信息；

(2) 日期和时间可以手动设置；

(3) 显示部分可以采用数码管或液晶模块。

附录A　ASCII表

Bin(二进制)	Dec(十进制)	Hex(十六进制)	缩写/字符	解　释
00000000	0	00	NUL(null)	空字符
00000001	1	01	SOH(startofheadline)	标题开始
00000010	2	02	STX(startoftext)	正文开始
00000011	3	03	ETX(endoftext)	正文结束
00000100	4	04	EOT(endoftransmission)	传输结束
00000101	5	05	ENQ(enquiry)	请求
00000110	6	06	ACK(acknowledge)	收到通知
00000111	7	07	BEL(bell)	响铃
00001000	8	08	BS(backspace)	退格
00001001	9	09	HT(horizontaltab)	水平制表符
00001010	10	0A	LF(NLlinefeed,newline)	换行键
00001011	11	0B	VT(verticaltab)	垂直制表符
00001100	12	0C	FF(NPformfeed,newpage)	换页键
00001101	13	0D	CR(carriagereturn)	回车键
00001110	14	0E	SO(shiftout)	不用切换
00001111	15	0F	SI(shiftin)	启用切换
00010000	16	10	DLE(datalinkescape)	数据链路转义
00010001	17	11	DC1(devicecontrol1)	设备控制1
00010010	18	12	DC2(devicecontrol2)	设备控制2
00010011	19	13	DC3(devicecontrol3)	设备控制3
00010100	20	14	DC4(devicecontrol4)	设备控制4
00010101	21	15	NAK(negativeacknowledge)	拒绝接收
00010110	22	16	SYN(synchronousidle)	同步空闲
00010111	23	17	ETB(endoftrans.block)	传输块结束
00011000	24	18	CAN(cancel)	取消
00011001	25	19	EM(endofmedium)	介质中断
00011010	26	1A	SUB(substitute)	替补
00011011	27	1B	ESC(escape)	换码(溢出)

续表一

Bin(二进制)	Dec(十进制)	Hex(十六进制)	缩写/字符	解释
00011100	28	1C	FS(fileseparator)	文件分割符
00011101	29	1D	GS(groupseparator)	分组符
00011110	30	1E	RS(recordseparator)	记录分离符
00011111	31	1F	US(unitseparator)	单元分隔符
00100000	32	20	(space)	空格
00100001	33	21	!	
00100010	34	22	"	
00100011	35	23	#	
00100100	36	24	$	
00100101	37	25	%	
00100110	38	26	&	
00100111	39	27	'	
00101000	40	28	(	
00101001	41	29	)	
00101010	42	2A	*	
00101011	43	2B	+	
00101100	44	2C	,	
00101101	45	2D	-	
00101110	46	2E	.	
00101111	47	2F	/	
00110000	48	30	0	
00110001	49	31	1	
00110010	50	32	2	
00110011	51	33	3	
00110100	52	34	4	
00110101	53	35	5	
00110110	54	36	6	
00110111	55	37	7	
00111000	56	38	8	
00111001	57	39	9	
00111010	58	3A	:	
00111011	59	3B	;	
00111100	60	3C	<	

续表二

Bin(二进制)	Dec(十进制)	Hex(十六进制)	缩写/字符	解　释
00111101	61	3D	=	
00111110	62	3E	>	
00111111	63	3F	?	
01000000	64	40	@	
01000001	65	41	A	
01000010	66	42	B	
01000011	67	43	C	
01000100	68	44	D	
01000101	69	45	E	
01000110	70	46	F	
01000111	71	47	G	
01001000	72	48	H	
01001001	73	49	I	
01001010	74	4A	J	
01001011	75	4B	K	
01001100	76	4C	L	
01001101	77	4D	M	
01001110	78	4E	N	
01001111	79	4F	O	
01010000	80	50	P	
01010001	81	51	Q	
01010010	82	52	R	
01010011	83	53	S	
01010100	84	54	T	
01010101	85	55	U	
01010110	86	56	V	
01010111	87	57	W	
01011000	88	58	X	
01011001	89	59	Y	
01011010	90	5A	Z	
01011011	91	5B	[	
01011100	92	5C	\	
01011101	93	5D	]	

续表三

Bin(二进制)	Dec(十进制)	Hex(十六进制)	缩写/字符	解　释
01011110	94	5E	^	
01011111	95	5F	_	
01100000	96	60	`	
01100001	97	61	a	
01100010	98	62	b	
01100011	99	63	c	
01100100	100	64	d	
01100101	101	65	e	
01100110	102	66	f	
01100111	103	67	g	
01101000	104	68	h	
01101001	105	69	i	
01101010	106	6A	j	
01101011	107	6B	k	
01101100	108	6C	l	
01101101	109	6D	m	
01101110	110	6E	n	
01101111	111	6F	o	
01110000	112	70	p	
01110001	113	71	q	
01110010	114	72	r	
01110011	115	73	s	
01110100	116	74	t	
01110101	117	75	u	
01110110	118	76	v	
01110111	119	77	w	
01111000	120	78	x	
01111001	121	79	y	
01111010	122	7A	z	
01111011	123	7B	{	
01111100	124	7C	|	
01111101	125	7D	}	
01111110	126	7E	~	
01111111	127	7F	DEL(delete)	删除

附录 B MCS-51 指令表

助记符	代码(十六进制)	说 明	字节	机器周期
数据传送指令				
* MOV A,Rn	E8~EF	工作寄存器送 A	1	1
* MOV A,direct	E5 direct	直接字节送 A	2	1
* MOV A,@Ri	E6~E7	间接 RAM 送 A	1	1
* MOV A,# data	74 data	立即数送 A	2	1
* MOV Rn,A	F8~FF	A 送工作寄存器	1	1
* MOV Rn,direct	A8~AF	直接字节送工作寄存器	2	2
* MOV Rn,# data	78~7F data	立即数送寄存器	2	1
* MOV direct,A	F5 direct	A 送直接字节	2	1
* MOV direct,Rn	88~8F direct	工作寄存器送直接字节	2	2
* MOV direct1,direct2	85 direct1 direct2	直接字节送直接字节	3	2
MOVdirect,@Ri	86~87	间接 RAM 送直接字节	2	2
* MOV direct,# data	75 direct data	立即数送直接字节	3	2
* MOV @Ri,A	F6~F7	A 送间接 RAM	1	1
MOV @Ri,direct	A6~A7 data	直接字节送间接 RAM	2	2
MOV @Ri,# data	76~77 data	立即数送间接 RAM	2	1
* MOV DPTR,# data16	90 data16	16 位常数送数据指针	3	2
* MOVC A,@A+DPTR	93	由((A)+(DPTR))寻址的程序存储器字节送 A	1	2
MOVC A,@A+PC	83	由((A)+(PC))寻址的程序存储器字节送 A	1	2
* MOVX A,@Ri	E2~E3	外部数据存储器(8 位地址寻址)送 A	1	2
* MOVX A,@DPTR	E0	外部数据存储器(16 位地址寻址)送 A	1	2
* MOVX @Ri,A	F2~F3	A 送外部数据存储器(8 位地址寻址)	1	2
* MOVX @DPTR,A	F0	A 送外部数据存储器(16 位地址寻址)	1	2
* PUSH direct	C0 direct	SP 加 1,直接字节进栈	2	2
* POP direct	D0 direct	直接字节退栈,SP 减 1	2	2

续表一

助记符	代码(十六进制)	说　明	字节	机器周期
* XCH A,Rn	C8～CF	交换 A 和工作寄存器	1	1
* XCH A,direct	C5 direct	交换 A 和直接字节	2	2
XCH A,@Ri	C6～C7	交换 A 和间接 RAM	1	1
XCHD A,@Ri	D6～D7	交换 A 和间接 RAM 的低位	1	1
* SWAP A	C4	A 高半字节和低半字节交换	1	1
算术运算指令				
* ADD A,Rn	28～2F	工作寄存器加到 A	1	1
* ADD A,direct	25 direct	直接字节加到 A	2	1
* ADD A,@Ri	26～27	间接 RAM 加到 A	1	1
* ADD A,#data	24 # data	立即数加到 A	2	1
* ADDC A,Rn	38～3F	工作寄存器和进位位加到 A	1	1
* ADDC A,direct	35 direct	直接字节和进位位加到 A	2	1
* ADDC A,@Ri	36～37	间接字节和进位位加到 A	1	1
* ADDC A,#data	34 data	立即数和进位位加到 A	2	1
* SUBB A,Rn	98～9F	A 减去工作寄存器和进位位	1	1
* SUBB A,direct	95 direct	A 减去直接字节和进位位	2	1
* SUBB A,@Ri	96～97	A 减去间接 RAM 和进位位	1	1
* SUBB A,#data	94 data	A 减去立即数和进位位	2	1
* INC A	04	A 加 1	1	1
* INC Rn	08～0F	工作寄存器加 1	1	1
INC direct	05 direct	直接字节加 1	2	1
* INC @Ri	06～07	间接 RAM 加 1	1	1
* DEC A	14	A 减 1	1	1
* DEC Rn	18～1F	工作寄存器减 1	1	1
DEC direct	15 direct	直接字节减 1	2	1
DEC @Ri	16～17	间接 RAM 减 1	1	1
* INC DPTR	A3	数据指针加 1	1	2
* MUL AB	A4	A 乘以 B	1	4
* DIV AB	84	A 除以 B	1	4
* DA A	D4	A 的十进制加法调整	1	1

续表二

助记符	代码(十六进制)	说　明	字节	机器周期
逻辑运算指令				
* ANL A,Rn	58～5F	工作寄存器“与”到 A	1	1
* ANL A,direct	55 direct	直接字节“与”到 A	2	1
* ANL A,@Ri	56～57	间接 RAM“与”到 A	1	1
* ANL A,# data	54 data	立即数“与”到 A	2	1
ANL direct, A	52 direct	A“与”到直接字节	2	1
ANL direct,# data	53 direct data	立即数“与”到直接字节	3	2
* ORL A,Rn	48～4F	工作寄存器“或”到 A	1	1
* ORL A,direct	45 direct	直接字节“或”到 A	2	1
* ORL A,@Ri	46～47	间接 RAM“或”到 A	1	1
* ORL A,# data	44 data	立即数“或”到 A	2	1
ORL direct, A	42direct	A“或”到直接字节	2	1
ORL direct,# data	43 direct data	立即数“或”到直接字节	3	2
* XRL A,Rn	68～6F	工作寄存器“异或”到 A	1	1
* XRL A,direct	65 direct	直接字节“异或”到 A	2	1
* XRL A,@Ri	66～67	间接 RAM“异或”到 A	1	1
* XRL A,# data	64 data	立即数“异或”到 A	2	1
XRL direct A	62 direct	A“异或”到直接字节	2	1
XRL direct,# data	63 direct data	立即数“异或”到直接字节	3	2
* CLR A	E4	A 清 0	1	1
* CPL A	F4	A 取反	1	1
* RL A	23	A 左环移	1	1
* RLC A	33	A 通过进位左环移	1	1
* RR A	03	A 右环移	1	1
* RRC A	13	A 通过进位右环移	1	1
控制转移指令				
ACALL addr11	xxx10001 addr(a7～a0)	绝对子程序调用	2	2
* LCALL addr16	12 addr(15～8) addr(7～0)	长子程序调用	3	2
* RET	22	子程序调用返回	1	2
* RETI	32	中断调用返回	1	2

续表三

助记符	代码(十六进制)	说明	字节	机器周期
AJMP addr11	xxx00001 addr(a7～a0)	绝对转移	2	2
* LJMP addr16	02 addr(15～8) addr(7～0)	长转移	3	2
SJMP rel	80 rel	短转移，相对转移	2	2
* JMP @A+DPTR	73	相对于 DPTR 间接转移	1	2
* JZ rel	60 rel	A 为 0 转移	2	2
* JNZ rel	70 rel	A 不为 0 转移	2	2
* CJNE A,direct,rel	B5 direct rel	直接字节与 A 比较，不等则转移	3	2
* CJNE A,#data,rel	B4 data rel	立即数与 A 比较，不等则转移	3	2
CJNE Rn,#data,rel	B8～BF data rel	立即数与工作寄存器比较，不等则转移	3	2
CJNE @Ri,#data,rel	B6～B7 data rel	立即数与间接 RAM 比较，不等则转移	3	2
* DJNZ Rn,rel	D8～DF rel	工作寄存器减 1，不为 0 则转移	2	2
DJNZ direct,rel	B5 direct rel	直接字节减 1，不为 0 则转移	3	2
* NOP	00	空操作	1	1
布尔变量操作指令				
* CLR C	C3	清 0 进位	1	1
* CLR bit	C2 bit	清 0 直接位	2	1
* SETB C	D3	置位进位	1	1
* SETB bit	D2 bit	置位直接位	2	1
* CPL C	B3	进位取反	1	1
* CPL bit	B2 bit	直接位取反	2	1
* ANL C,bit	82 bit	直接数“与”到进位	2	2
ANL C,/bit	B0 bit	直接位的反“与”到进位	2	2
* ORL C,bit	72 bit	直接位“或”到进位	2	2
ORL C,/bit	A0 bit	直接位的反“或”到进位	2	2
* MOV C,bit	A2 bit	直接位送进位	2	1
* MOV bit,C	92 bit	进位送直接位	2	2
* JC rel	40 rel	进位位为 1 转移	2	2
* JNC rel	50 rel	进位位为 0 转移	2	2
* JB bit,rel	20 bit rel	直接位为 1 转移	3	2
* JNB bit,rel	30 bit rel	直接位为 0 转移	3	2
* JBC bit,rel	10 bit rel	直接位为 1 转移，然后清 0 该位	3	2

注：带有 * 号的指令是常用指令。

参考文献

[1] 李桂林,马驰,王新屏,等. 单片机原理及应用[M]. 北京:电子工业出版社,2012.

[2] 李桂林. 单片机原理与应用开发教程[M]. 北京:电子工业出版社,2016.

[3] 李全利. 单片机原理及接口技术[M]. 2版. 北京:高等教育出版社,2009.

[4] 张毅刚,彭喜元,姜守达,等. 新编MCS-51单片机应用设计[M]. 3版. 哈尔滨:哈尔滨工业大学出版社,2008.

[5] 潘明,黄继业,潘松. 单片机原理与应用技术[M]. 北京:清华大学出版社,2011.

[6] 胡汉才. 单片机原理及其接口技术[M]. 3版. 北京:清华大学出版社,2010.

[7] 林立. 单片机原理及应用:基于Proteus和Keil C[M]. 北京:电子工业出版社,2009.

[8]张毅刚,王少军,付宁. 单片机原理及接口技术[M]. 2版. 北京:人民邮电出版社,2015.

[9] 李全利. 单片机原理及应用[M]. 2版. 北京:清华大学出版社,2014.

[10] 张毅刚,赵光权,张京超. 单片机原理及应用:C51编程+Proteus仿真[M]. 2版. 北京:高等教育出版社,2016.

[11] 魏立峰,王宝兴. 单片机原理与应用技术[M]. 北京:北京大学出版社,2006.

[12] 楼然苗,胡佳文,李光飞,等. 51系列单片机原理及设计实例[M]. 北京:北京航空航天大学出版社,2010.

[13] 马秀丽,周越,王红. 单片机原理与应用系统设计[M]. 北京:清华大学出版社,2014.



单片机原理及应用

本书以MCS－51系列单片机原理和应用为主线，重点介绍单片机的结构、指令系统、汇编程序设计、内部标准功能单元、硬件系统扩展等内容，并精心设计了大量例题和多种解题思路，启发学生学习。

本书既注重基础知识的讲解和逻辑思维的训练，又突出工程实践和实际应用。

本书既可作为相关专业的教材，也可作为电子设计、开发爱好者的参考书。

作者精心开发了一个极具特色的、内容丰富的网站（同时支持手机版），可供读者参考学习。



下载手机App，提供知识点、例题及动画方式讲解的全新体验


ISBN 978-7-5606-4592-6
9 787560 645926 >
定价：34.00元